Capítulo IV: UML estructural

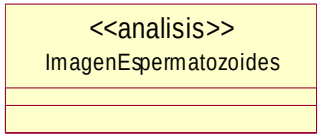
carlos.platero@upm.es

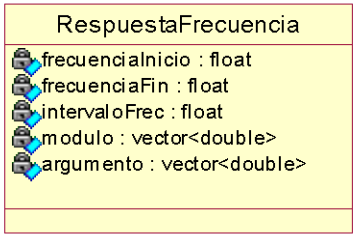
UML (*Unified Modeling Language*)

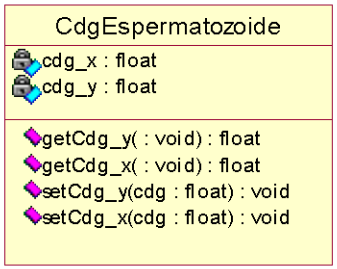
- Vistas (artefactos que representan un aspecto del sistema)
  - Estructural: estructura lógica: clases, paquetes y casos de uso (Diagramas de clase y de casos de uso)
  - Dinámico: interacciones entre objetos (Diagramas de interacción, de actividades y máquinas de estado)
  - Implementación: componentes (diagramas de componentes y despliegue)
- OMG y UML

Clases en UML

- Clases en UML: rectángulo con tres compartimentos
  - Nombre de la clase (Sustantivo y empieza por Mayúscula)
  - Atributos (identificador y tipo; sustantivo y empieza por minúscula)
    - **Visibilidad nombre_atributo ':' tipo_atributo '=' valor inicial ´{`otras propiedades '}'**
  - Servicios (frase verbal y empieza por minúscula)
    - **Visibilidad nombre_servicio '('lista de parámetros')':'tipo de retorno' {`otras propiedades´}`**







Ejemplo 4.1

- Realizar la implementación en C++ de la siguiente descripción UML referente a la clase Esfera.


```
#ifndef _INC_ESFERA_
#define _INC_ESFERA_

class Esfera
{
private:
        float radio_esfera;

public:
 Esfera()
        {this->radio_esfera = 2.0f;}
 float getRadio()
        {return (this->radio_esfera);}
 void setRadio(float radio)
        {this->radio_esfera=radio;}
};
#endif
```

Ejercicio

- Ingeniería directa

Ejercicio

```
#ifndef _DINERO_INC_
#define _DINERO_INC_

typedef enum {EURO, DOLAR, LIBRA} TipoDinero;

class Dinero
{
        TipoDinero elTipoMoneda;
        float cantidad;
public:
        Dinero(): elTipoMoneda(EURO), cantidad(0) {}
        Dinero(float valor, TipoDinero elTipo): elTipoMoneda(elTipo),cantidad(valor) {}

        Dinero(Dinero &elValor)
        {
                elTipoMoneda = elValor.elTipoMoneda;
                cantidad = elValor.cantidad;
        }

        Dinero& operator= (Dinero &elValor)
        {
                elTipoMoneda = elValor.elTipoMoneda;
                cantidad = elValor.cantidad;
                return(*this);
        }

        void setCantidad(float laCantidad){cantidad=laCantidad;}
        float getCantidad(void){return cantidad;}
        void setTipoDinero(TipoDinero elTipo){elTipoMoneda=elTipo;}
        TipoDinero getTipoDinero(void){return elTipoMoneda;}
};

#endif
```

Ingeniería inversa

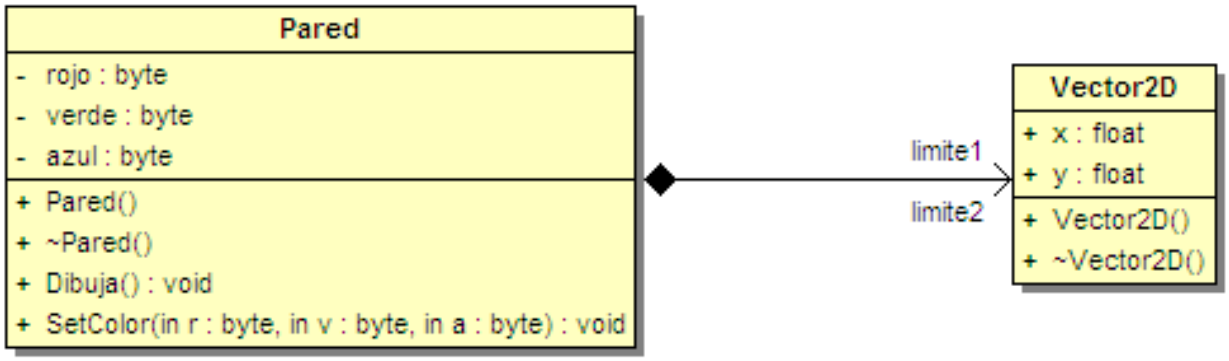
```
#if !defined(AFX_PARED)
#define AFX_PARED

#include "comun/Vector2D.h"

class Pared
{
public:
 Pared();
 virtual ~Pared();
 void Dibuja();
 void SetColor( unsigned char r,unsigned char v,
                unsigned char a);
 Vector2D limite1;
 Vector2D limite2;

private:
 unsigned char rojo;
 unsigned char verde;
 unsigned char azul;
};

#endif
```

```
#if !defined(AFX_VECTOR2D)
#define AFX_VECTOR2D


class Vector2D
{
public:
        Vector2D();
        virtual ~Vector2D();
        float x;
        float y;

};

#endif
```

Ingeniería inversa

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

#include "comun/Vector2D.h"

class Pared
{
public:
 Pared();
 virtual ~Pared();
 void Dibuja();
 void SetColor( unsigned char r,unsigned char v,
                unsigned char a);
 Vector2D limite1;
 Vector2D limite2;

private:
 unsigned char rojo;
 unsigned char verde;
 unsigned char azul;
};

#endif
```

```
#if !defined(AFX_VECTOR2D)
#define AFX_VECTOR2D
class Vector2D
{
public:
 Vector2D();
 virtual ~Vector2D();
 float x;
 float y;
};
#endif
```

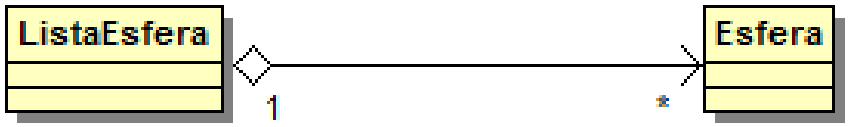


Tipos de clases

- Clases parametrizadas
- Interfaces

Clases parametrizadas

- Clases parametrizadas
  - Contenedores (templete)
  - Artefactos del SW
  - ANSI C++ (STL)
  - Java NO
  - Clases instanciadas



Opción 1: Esfera * lista[MAX_ESFERAS];
Opción 2: std::vector<Esfera *> lista;

Ejemplo de clase parametrizadas

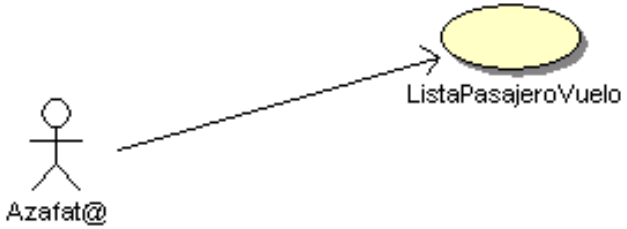
- Se desea realizar una aplicación para un *pocket* sobre los pasajeros de un vuelo. Plantéese bajo los *frameworks* de ANSI C++.

1. Características principales del sistema
   - El sistema debe de tener de cada pasajero, el nombre, el número de pasaporte y el asiento.
   - El sistema debe dar el número total de pasajeros, de asientos ocupados y asientos libres.
   - El sistema debe de listar los datos de todos los pasajeros.
   - El sistema debe de añadir datos sobre los pasajeros.
2. Los términos para el glosario serían:
   - Pasajero, Vuelo, Asiento, Pasaporte, Nombre,....
3. La lista de evento-actor-objetivo estaría constituida por:

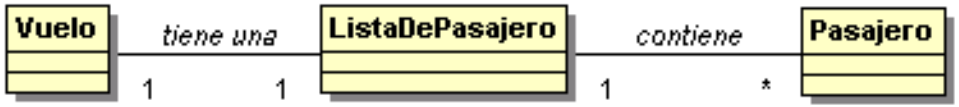
Evento	Actor	Objetivo
Introducir datos pasajero	Azafat@	Formar la base de datos del pasaje
Visualizar datos pasajero	Azafat@	Verificar los datos de un pasajero
Ocupación del vuelo	Azafat@	Obtener datos de ocupación del vuelo

Ejemplo de clase parametrizadas

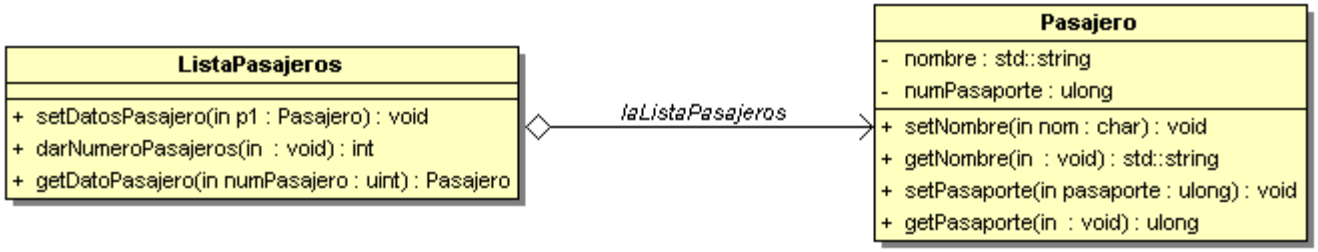
4. Casos de uso



5. AOO



6. DOO



Ejemplo de clase parametrizadas

7. Implementación

```
#ifndef _PASAJERO_INC_
#define _PASAJERO_INC_

#include <string>

class Pasajero
{
public:
    void setNombre(const char *nom)
    {nombre = nom;}
    std::string & getNombre( void )
        {return  nombre;}
    void setPasaporte(unsigned long pasaporte)
    {numPasaporte=pasaporte;}
    unsigned long getPasaporte( void ) const
    {return (numPasaporte);}

private:
    std::string nombre;
    unsigned long numPasaporte;
};

#endif
```

```
#ifndef _INC_LISTA_PASAJEROS
#define _INC_LISTA_PASAJEROS

#include <vector>
#include "Pasajero.h"

class ListaPasajeros
{
public:
    void setDatosPasajero (const Pasajero p1)
    {laListaPasajeros.push_back(p1);}
    int darNumeroPasajeros ( void ) const
    { return (laListaPasajeros.size()); }
    Pasajero  getDatoPasajero( unsigned numPasajero )
    { return laListaPasajeros[numPasajero];}


private:
    std::vector<Pasajero> laListaPasajeros;
};

#endif
```

Problema para casa

- Cambiar Esfera * lista[MAX_ESFERAS]; por std::vector<Esfera *> lista; en la clase ListaEsferas

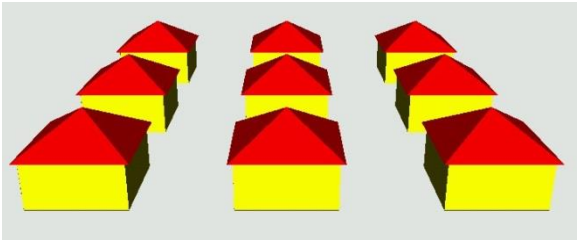
```
#include "Esfera.h"
#include <vector>
//#define MAX_ESFERAS   100
class ListaEsferas
{
    friend class Interaccion;
    int numero;
    //Esfera * lista[MAX_ESFERAS];
    std::vector<Esfera *> lista;
public:
    void DestruirContenido();
    void Mueve(float );
    void Dibuja();
    bool agregarEsfera(Esfera *);
    void EliminarEsfera(int ind);
    void EliminarEsfera(Esfera *e);
    bool colision(Esfera &e1,Esfera &e2);
    Esfera * ListaEsferas::Colision(Esfera &e);
    Esfera * operator [](int i);
    int getNumero () {return numero;}

    ListaEsferas();
    virtual ~ListaEsferas();
};
```

Examen: Urbanización

Se desea hacer una representación grafica como la de la figura de una urbanización formada por filas y columnas. Para ello se han desarrollado ya las clases **Techo** y **Bloque** que tienen las siguientes definiciones, y que pueden ser utilizadas como se indica en el *main()*. Cada casa estará formada por una única planta definida por un bloque y un techo. Se pide:

1. Representación UML de las clases Techo y Bloque. (1.5 puntos)
2. Diagrama de Clases de Diseño (DCD) de la solución. (3 puntos)
3. Diagrama de Secuencias explicativo del dibujo de la escena.

   (2.5 puntos)
4. Implementación C++ (excluidas clases Techo y Bloque, incluido main). (3 puntos)



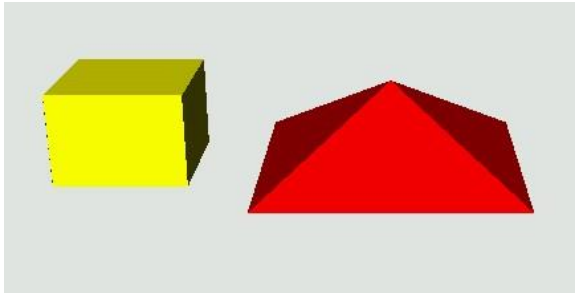
El ancho de las casas es de 1 unidad, su altura de 0.5 y la separación entre casas es de 3 unidades.

Examen: Urbanización

```
Class Techo
{
public:
 void Dibuja();
 void SetPos(float xp,float yp,
             float zp);
 void SetBase(float b);

protected:
          float x,y,z;
          float base;
};
```

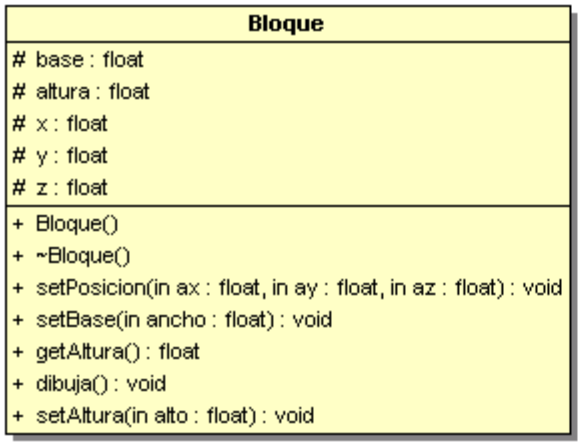
```
class Bloque
{
public:
 float GetAltura();
 void SetAltura(float a);
 void SetBase(float b);
 void SetPos(float px, float py,
             float pz);
 void Dibuja();

protected:
          float x,y,z;
          float base;
          float altura;
};
```

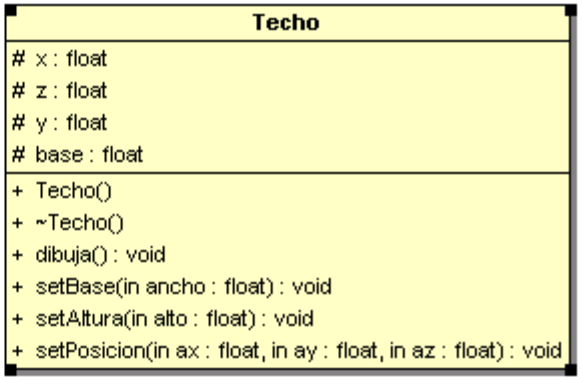
```
void main()
{
 Techo t;
 Bloque b;
 t.SetPos(0,0,0);
 t.SetBase(2);
 b.SetPos(2,0,0);
 b.SetAltura(0.7);
 b.SetBase(1);
 t.Dibuja();
 b.Dibuja();
}
```



Examen: Urbanización (1)

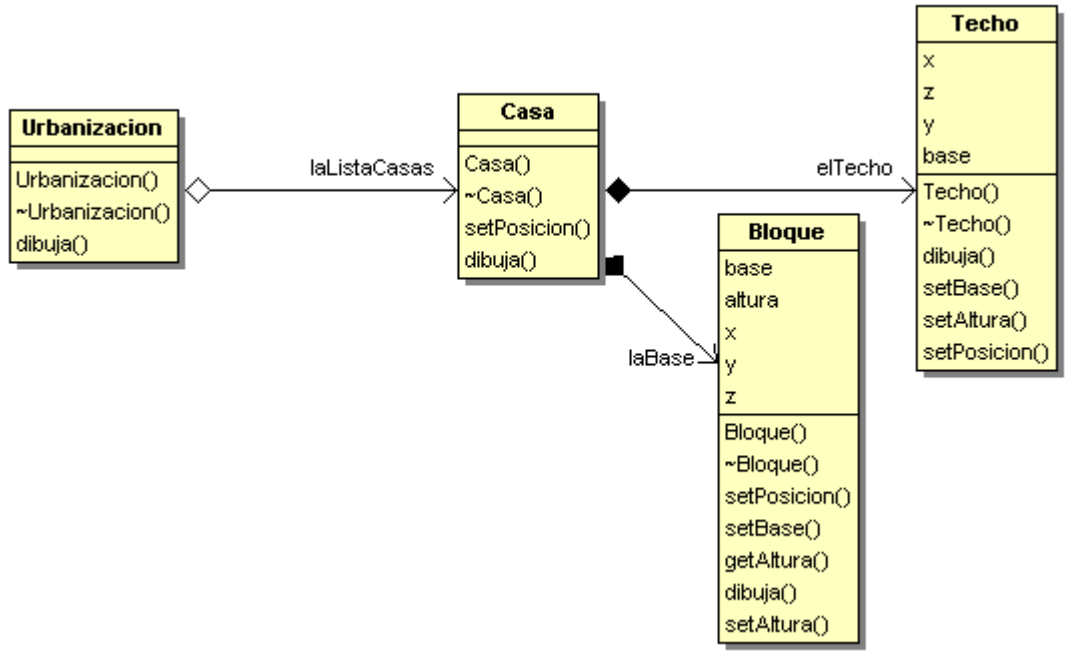
1. Representación UML de las clases Techo y Bloque.





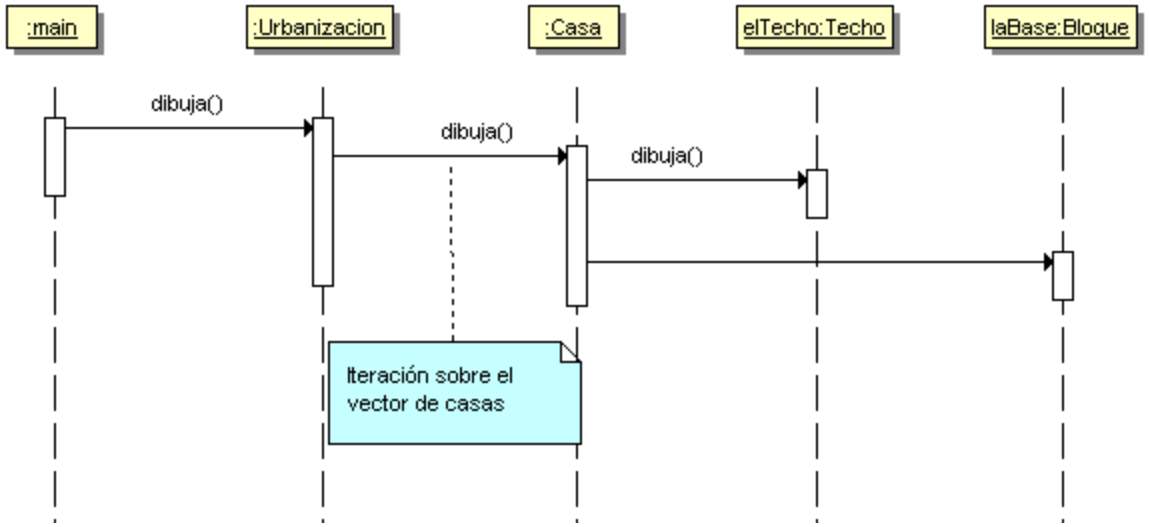
Examen: Urbanización (2)

2. Diagrama de Clases de Diseño (DCD) de la solución.



Examen: Urbanización (3)

3. Diagrama de Secuencias explicativo del dibujo de la escena.



Examen: Urbanización (4)

4. Implementación en C++

```
#include "Bloque.h"
#include "Techo.h"

class Casa
{
		Bloque laBase;
		Techo elTecho;

public:
		Casa(float,float,float);
		virtual ~Casa();
		void setPosicion(float,float,float);
		void dibuja();

};
```

```
Casa::Casa(float ancho, float altoBase, float altoTejado)
{
		this->laBase.setBase(ancho);
		this->elTecho.setBase(ancho);

		this->laBase.setAltura(altoBase);
		this->elTecho.setAltura(altoTejado);

}

Casa::~Casa()
{

}

void Casa::setPosicion(float ax,float ay,float az)
{
		this->laBase.setPosicion(ax,ay,az);
		this->elTecho.setPosicion(ax,ay+(this->laBase.getAltura()),az);
}

void Casa::dibuja()
{
		this->elTecho.dibuja();
		this->laBase.dibuja();
}
```

Examen: Urbanización (5)

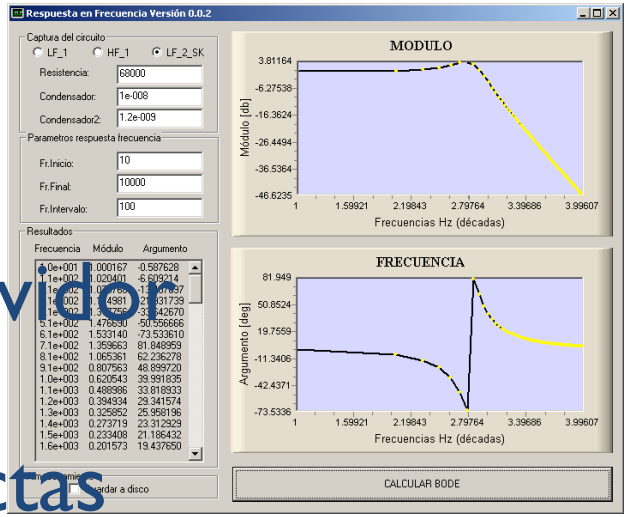
4. Implementación en C++

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

class Urbanizacion
{
        unsigned filasCasa, columnasCasa;
        std::vector<Casa> laListaCasas;

public:

        Urbanizacion(unsigned, unsigned);
        virtual ~Urbanizacion();
        void dibuja();

};
```

```
#define ANCHO_BLOQUE        1.0f
#define ALTO_BLOQUE         0.5f
#define ANCHO_TECHO         1.0f
#define SEPARACION_CASA     3.0f

Urbanizacion::Urbanizacion(unsigned filas, unsigned columnas)
{
 filasCasa = filas; columnasCasa = columnas;
 for(unsigned i = 0; i<filas; i++)
    for(unsigned j = 0; j<columnas; j++){
        this->laListaCasas.push_back
        (Casa(ANCHO_BLOQUE,ALTO_BLOQUE,ANCHO_TECHO));
        this->laListaCasas[(i*columnas)+j].setPosicion
        (SEPARACION_CASA*j,0,SEPARACION_CASA*i);
        }
}

Urbanizacion::~Urbanizacion()
{

}

void Urbanizacion::dibuja()
{
        for (unsigned i=0;i<filasCasa*columnasCasa; i++)
                this->laListaCasas[i].dibuja();
}
```

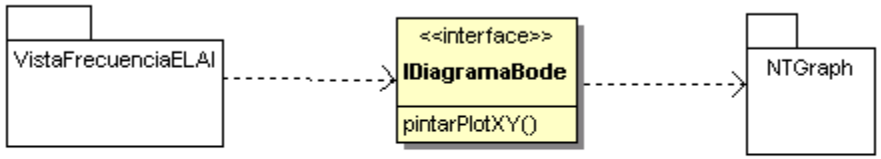
Interfaces

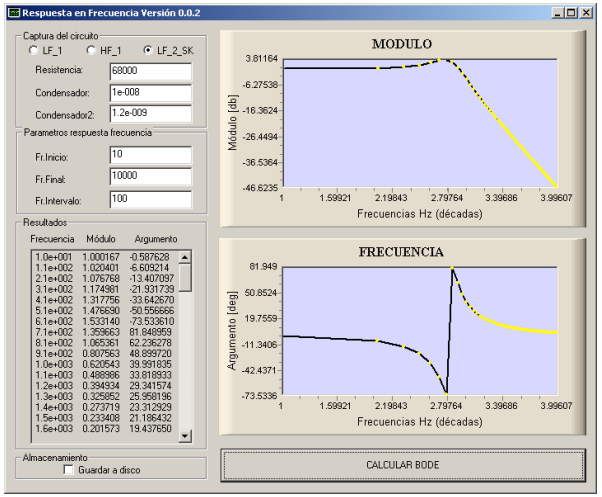
- Interfaces
  - Especifica algunos servicios públicos del paquete
  - Estereotipo <<interface>>
  - Carece de atributos
  - Relaciones entre cliente y servidor
  - “Cortafuegos”
  - C++ emplea las clases abstractas
  - Palabra clave interface (JAVA y C#)

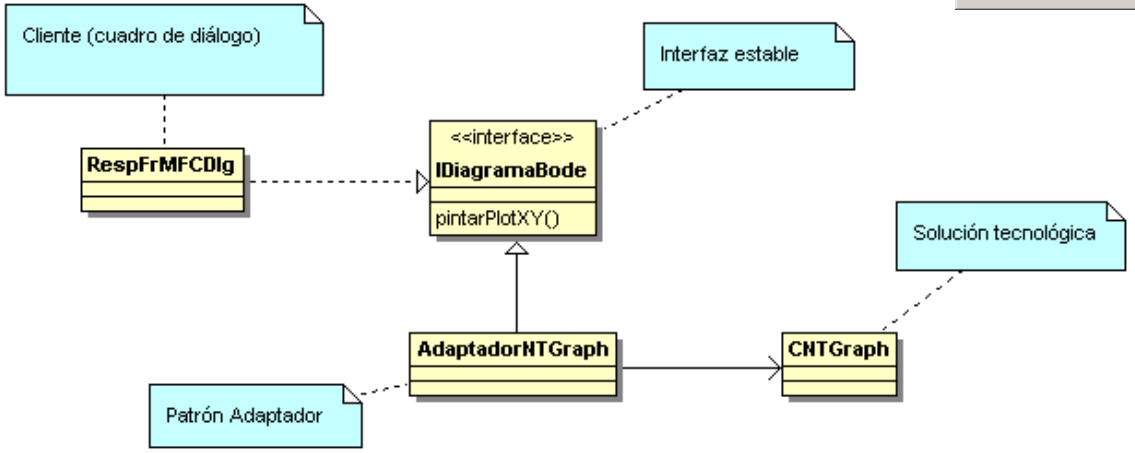


Ejemplo de interfaz

- Definir un interfaz para la visualización de los diagramas de Bode en la aplicación de respuesta en frecuencia de los filtro lineales.



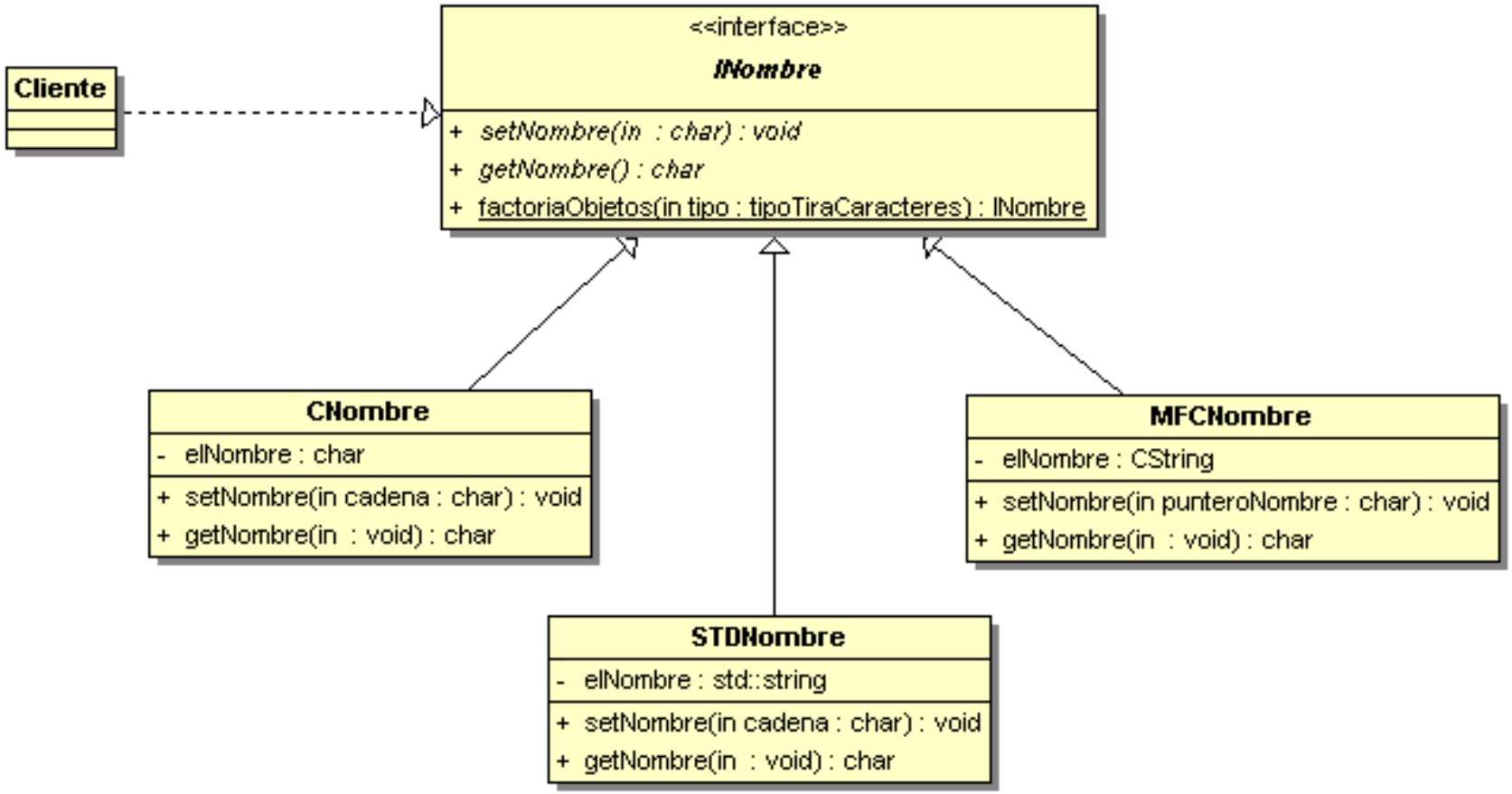




Ejemplo de interfaz (1)

```
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 Perez");

        std::cout << pNombre1->getNombre() << std::endl;
        std::cout << pNombre2->getNombre() << std::endl;
        std::cout << pNombre3->getNombre() << std::endl;


        delete pNombre1, pNombre2;
        delete pNombre3;

        return 0;

}
```

Ejemplo de interfaz (2)



Ejemplo de interfaz (3)

```
#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)
           { strcpy (elNombre, cadena); }
 virtual const char * getNombre (void)
           { return (elNombre);}
private:
          char elNombre[80];
};
#endif
```

```
#ifndef _INC_STDNOMBRE_
#define _INC_STDNOMBRE_

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

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;
};
#endif
```

Ejemplo de interfaz (4)

- 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");

            cout << pNombre1->getNombre() << endl;
            cout << pNombre2->getNombre() << endl;
            cout << pNombre3->getNombre() << endl;


            delete pNombre1, pNombre2, pNombre3;
            return 0;
}
```

Tipos de relaciones y asociaciones

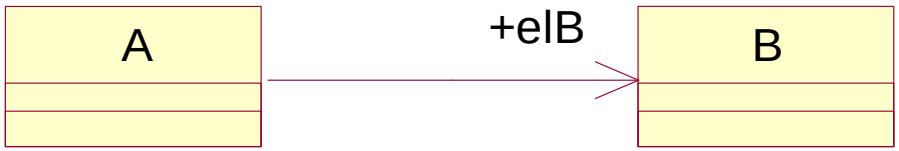
- Tipos de relaciones:
  - Asociaciones, generalizaciones y dependencias
- Asociaciones
  - Relaciones de necesito-conocer
  - Bidireccionales en AOO
    - relaciones entre categorías conceptuales
  - Unidireccionales en DOO
    - Caminos de visibilidad y navegabilidad
    - El más empleado en POO
    - Lazo mortal

Ejemplo de lazo mortal


```
class B;

class A
{
public:
          B elB;

};
```

```
class A;

class B
{
public:
          A elA;

};
```

Ejemplo de lazo mortal

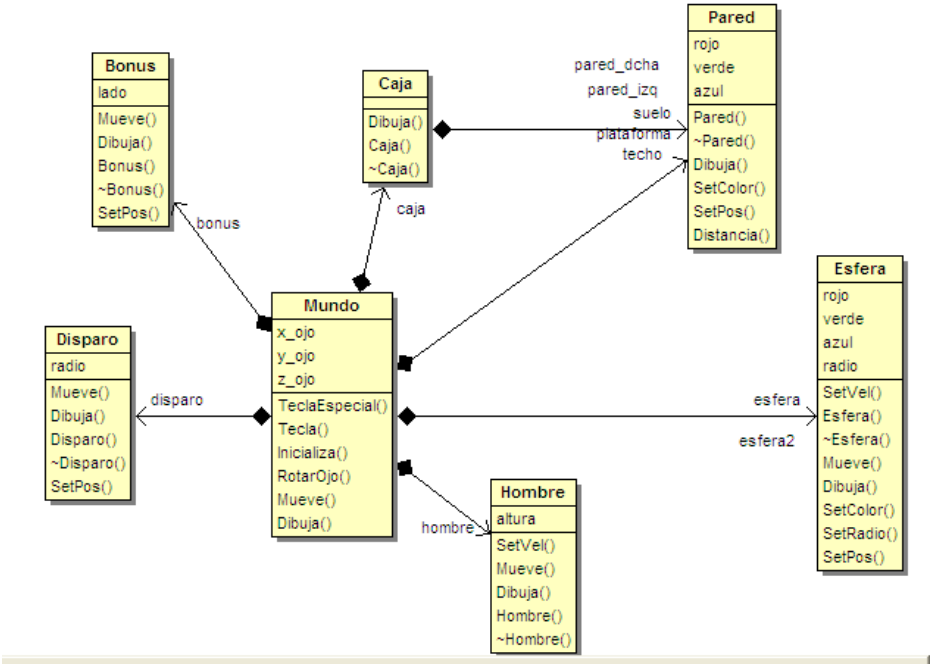



```
class B;

class A
{
public:
          B elB;

};
```

```
class B
{
};
```

Asociaciones

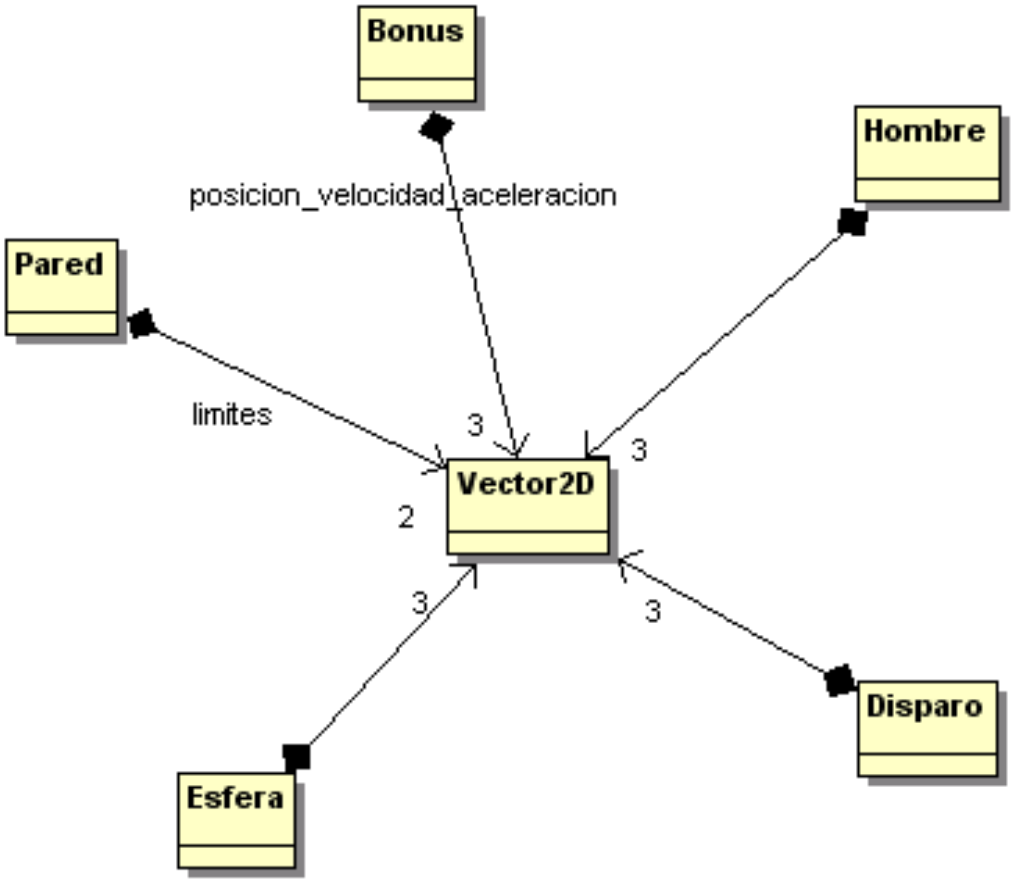
- Guía de las asociaciones en el DOO
  - Un objeto de la clase A usa un servicio de un objeto de la clase B.
  - Un objeto de la clase A crea un objeto de la clase B.
  - Un objeto de la clase A tiene un atributo cuyos valores son objetos de la clase B o colecciones de objetos de la clase B.
  - Un objeto de la clase A recibe un mensaje con un objeto de la clase B pasado como argumento.



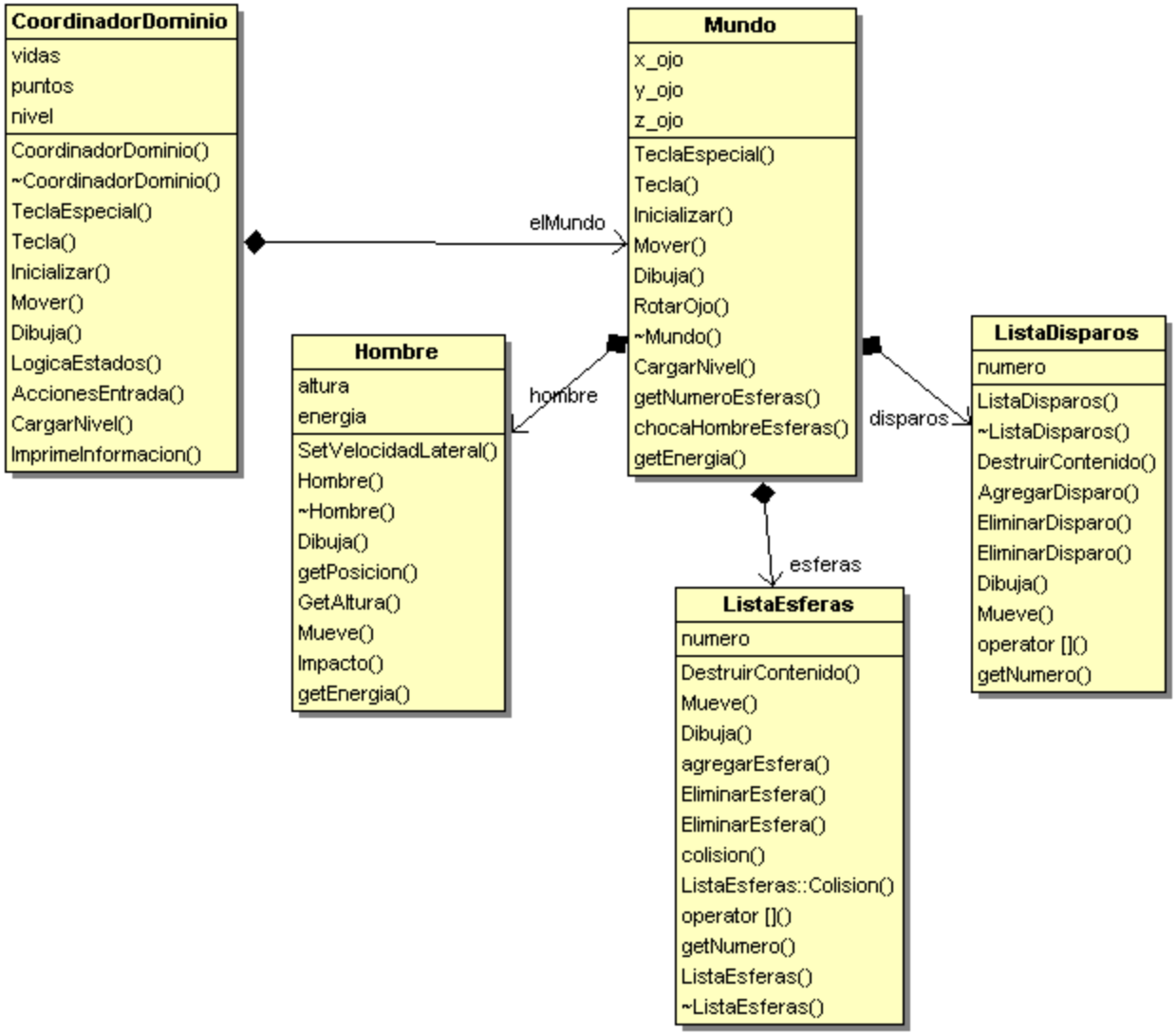
Agregación

- Agregación
  - Tipo particular de asociación, del todo con las partes
  - Agregación por valor o composición es una agregación unida con la creación
    - Las partes no se pueden intercambiar
  - Guía de asociación
    - Existe un ensamblaje obvio del todo con las parte.
    - Alguna propiedad del compuesto se propaga a las partes.
    - El tiempo de vida de la parte está unida al tiempo de vida del compuesto.
  - Beneficios
    - No en AOO
    - Las operaciones –como la copia y la eliminación– que se aplican al todo a menudo se propagan a las partes.

Ejemplo del Pang

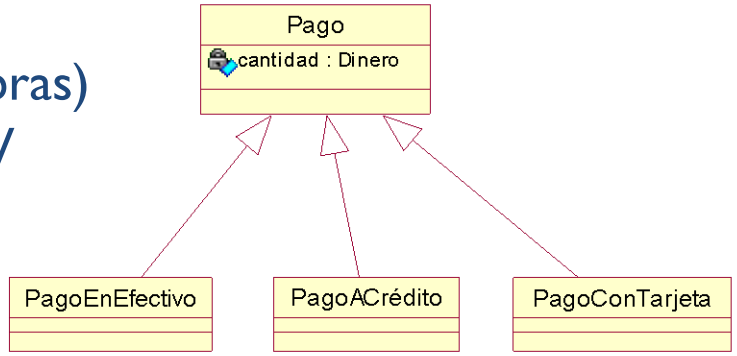


Ejemplo del Pang

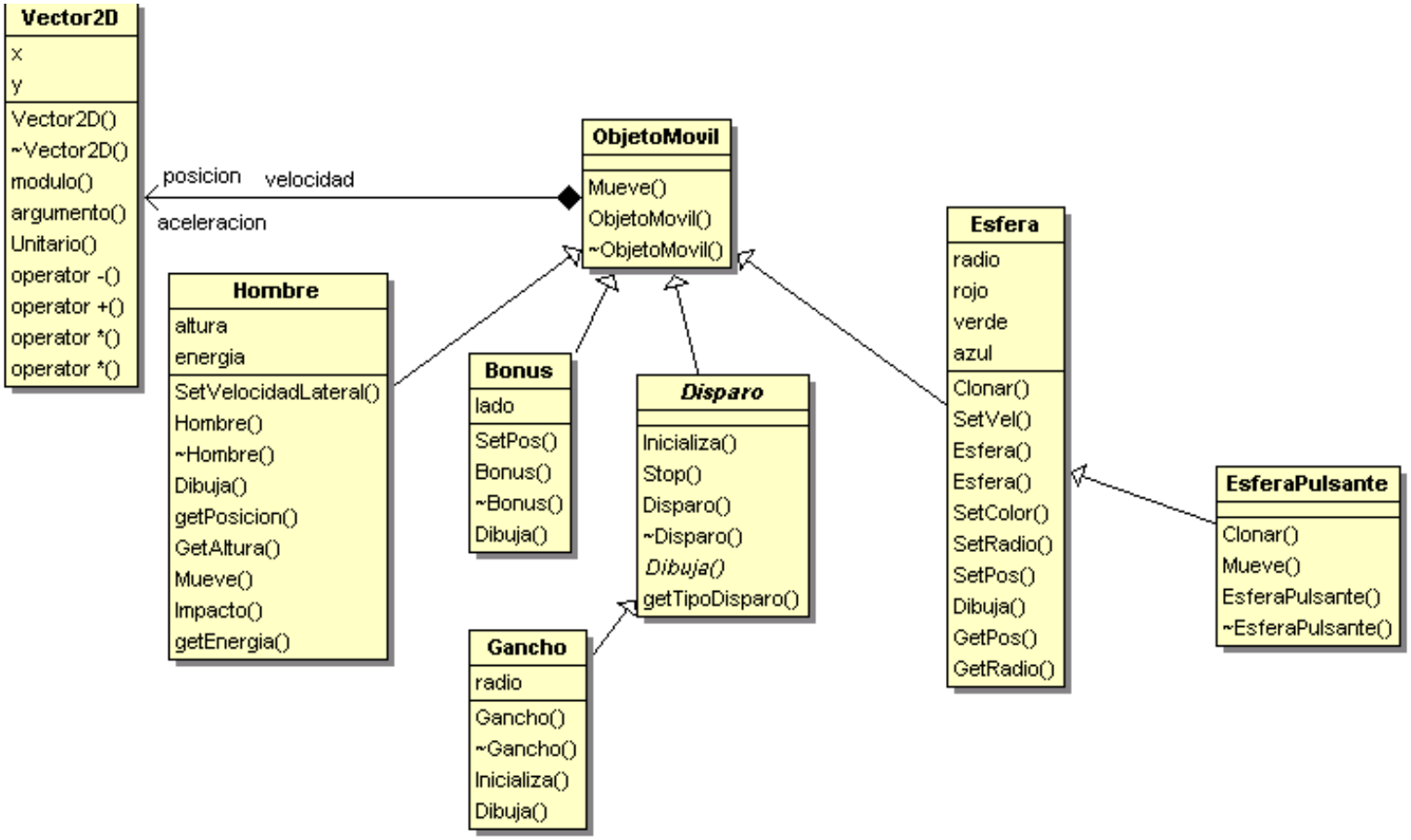


Generalización

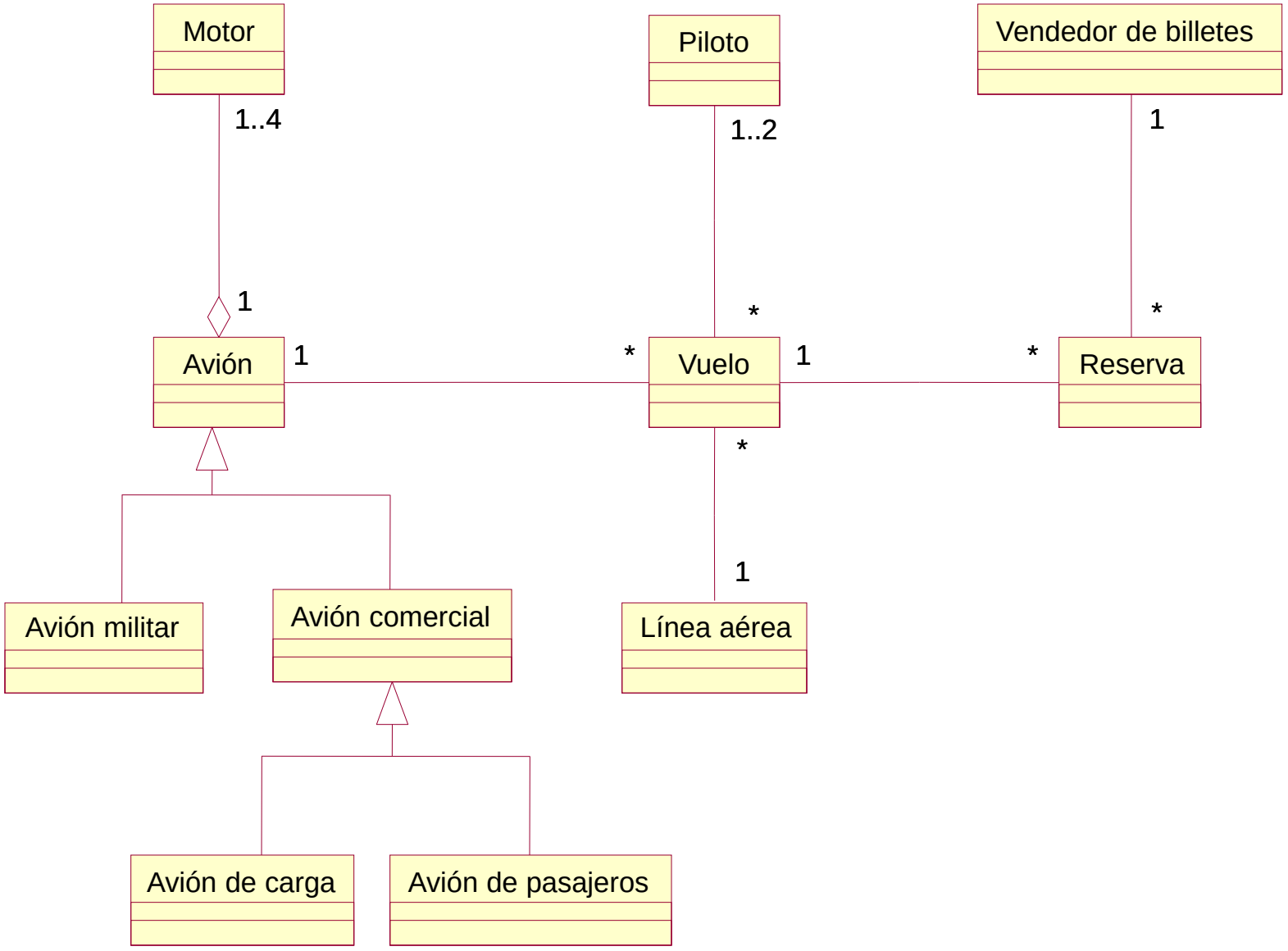
- Generalización
  - Factorizar conceptos (economía de palabras)
  - Inspiración para la jerarquía de clases SW
- Reglas
  - "100%" y "Es un"
- Subclases:
  - La subclase tiene atributos adicionales.
  - La subclase tiene asociaciones adicionales.
  - La subclase funciona de manera diferente e interesante a la superclase o a otras subclases.
- Superclase
  - Cuando las subclases potenciales representen variaciones de un concepto similar.
  - Las subclases se ajustarán a las reglas del 100% y Es-un.
  - Todas las subclases tienen el mismo atributo que se puede factorizar y expresarlo en la superclase.
  - Todas las subclases tienen la misma asociación que se puede factorizar y relacionar con la superclase.



Ejemplo del Pang



Ejercicio


Motor
Piloto
Vendedor de billetes
1..4
1..2
1
1
*
*
Avión
1
*
Vuelo
1
*
Reserva
*
1
Avión militar
Avión comercial
Línea aérea
Avión de carga
Avión de pasajeros

Generalización & Herencia

- Generalización
  - Factorizar conceptos (economía de palabras)
- Reglas
  - "100%" y "Es un"
- Herencia
  - Implementación de clases SW: Problemas - Composición
  - Sólo cuando procede de la jerarquía de clases conceptuales

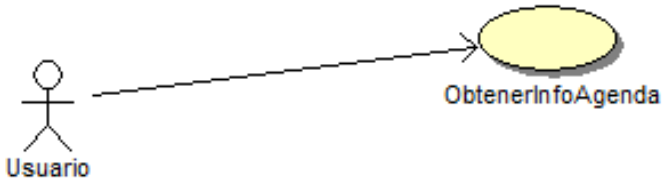
Ejemplo de agenda telefónica

Realizar una agenda telefónica

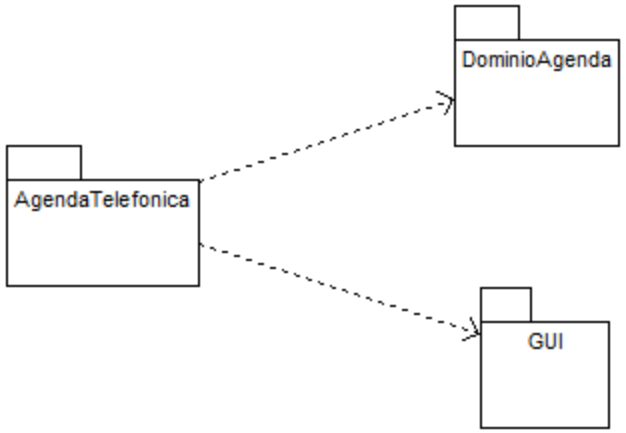
1. Jerarquía a dos niveles
   - Las características de la aplicación, en jerarquía a dos niveles, son:
     1. La agenda telefónica debe contener y gestionar la lista de teléfonos de los contactos:
        1.a. Debe permitir añadir, eliminar, listar un contacto
2. Glosario
   - El glosario estaría formado por: listín telefónico, contacto, nombre, teléfono, agenda telefónica,…

Ejemplo de agenda telefónica

3. El caso de uso sería *ObtenerInformaciónAgendaTelefónica*.

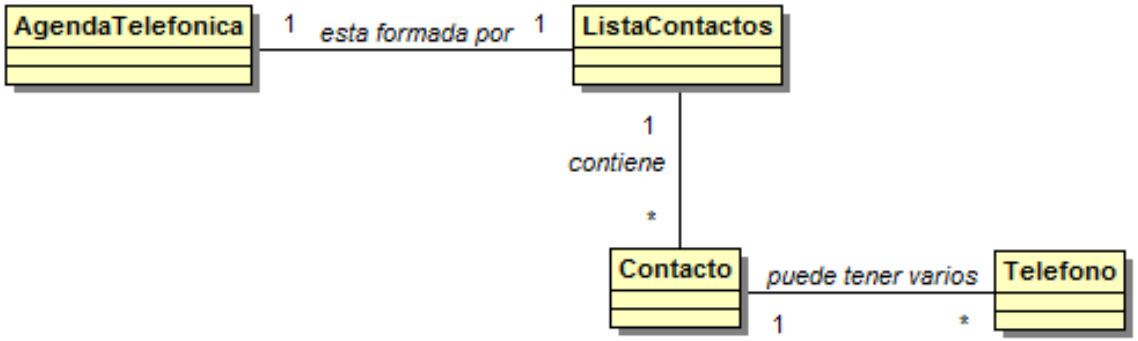


4. La arquitectura básica estaría formada por un paquete del dominio y otro de visualización.

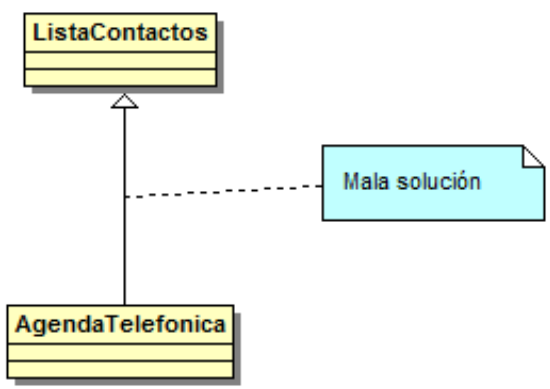


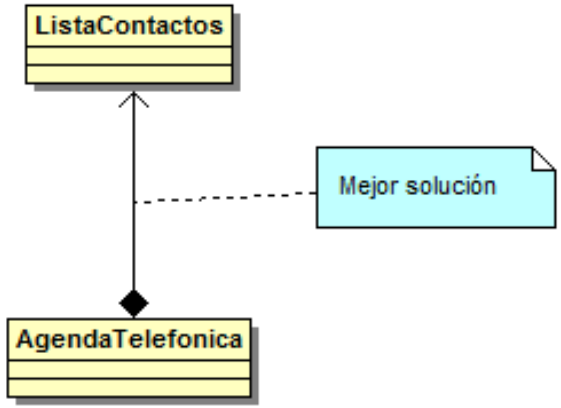
Ejemplo de agenda telefónica

3. Modelo del Dominio.:



4. Diseño:

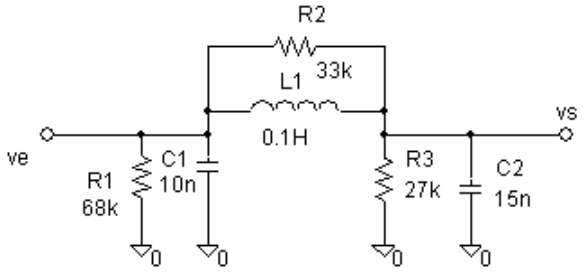




Examen de primer parcial (octubre 14)

Se pretende realizar un simulador de cuadripolos eléctricos pasivos. Estas redes sólo están formadas por la combinación de resistencias, condensadores y bobinas. En esta primera iteración se pretende guardar la información de los componentes y su listado.

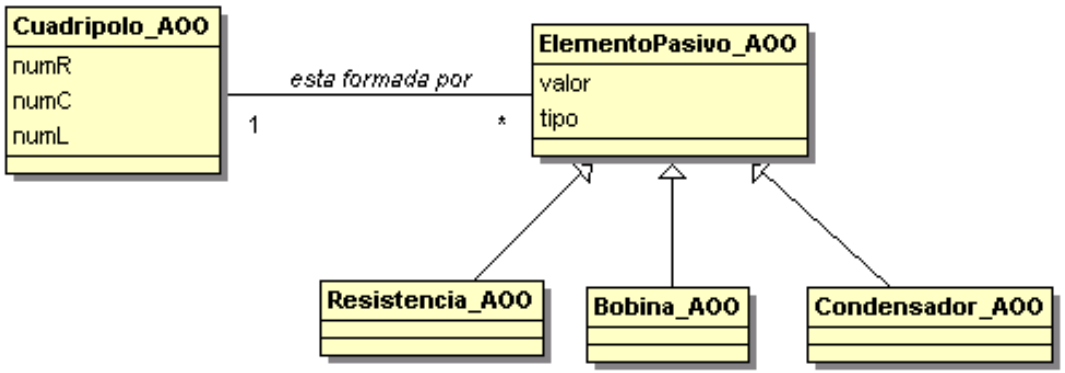
1. Modelo del dominio y diagrama de secuencia del sistema (2.5 puntos).
2. Diagrama de clases de diseño (2.5 puntos).
3. Implementación en C++ de la solución (5 puntos).

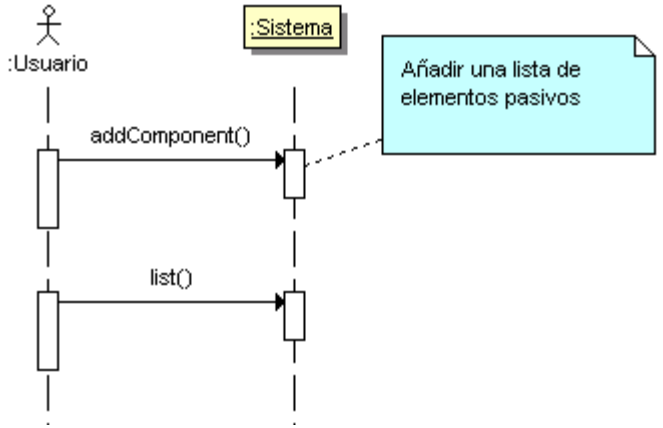



```
// Código de test
int main() {
Cuadripolo red;
red.addComponent(R,68e3);
red.addComponent(R,33e3);
red.addComponent(R,27e3);
red.addComponent(L,0.1);
red.addComponent(C,1e-8);
red.addComponent(C,1.5e-8);
red.list();
return 0;
}

////////////////////////////////////
//Resultado en consola
////////////////////////////////////
Resistencia 1: valor = 68000
Resistencia 2: valor = 33000
Resistencia 3: valor = 27000
Bobina 1: valor = 0.1
Condensador 1: valor = 1e-008
Condensador 2: valor = 1.5e-008
```

Examen de primer parcial (octubre 14)

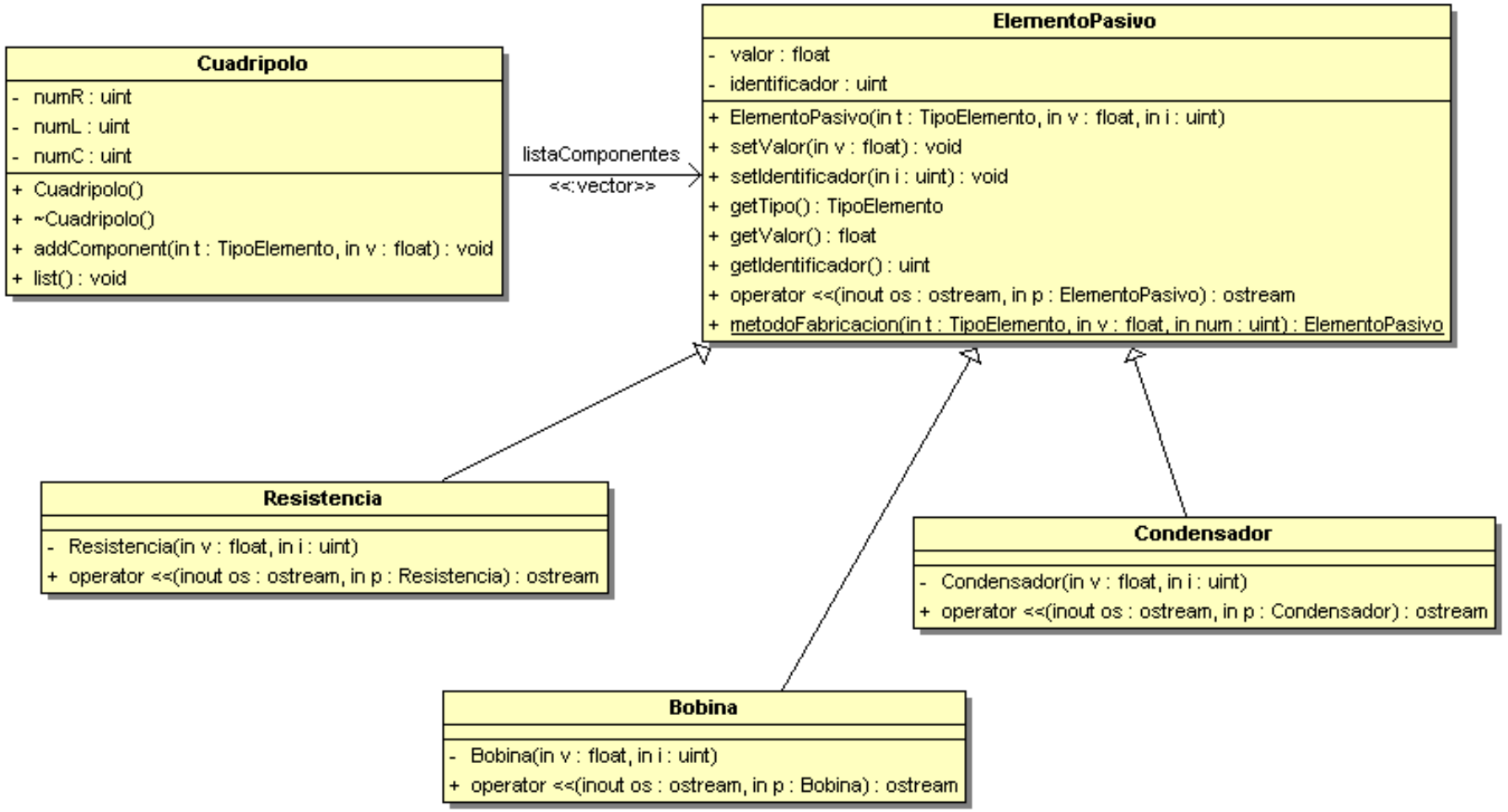
- Modelo del dominio y diagrama de secuencia del sistema (2.5 puntos).





Examen de primer parcial (octubre 14)

- Diagrama de clases de diseño (2.5 puntos).



Examen de primer parcial (octubre 14)

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

typedef enum{R,L,C} TipoElemento;

class ElementoPasivo {
          TipoElemento tipo;
          float valor;
          unsigned identificador;
public:
          ElementoPasivo(TipoElemento t,float v,unsigned i):
            tipo(t), valor(v),identificador(i) {}
          void setValor(float v) {valor=v;}
          void setIdentificador(unsigned i) {identificador=i;}
          TipoElemento getTipo() { return tipo;}
          float getValor() {return valor;}
          unsigned getIdentificador() {return identificador;}
          friend ostream& operator<<(ostream& os, const ElementoPasivo * p) {
                    return os << p->identificador << ": valor = " << p->valor << endl;
          }

          static ElementoPasivo * metodoFabricacion(TipoElemento,float,unsigned);
};
```

Examen de primer parcial (octubre 14)

```
class Resistencia: public ElementoPasivo {
        friend class ElementoPasivo;
        Resistencia(float v,unsigned i): ElementoPasivo(R,v,i) {}
public:
        friend ostream& operator<<(ostream& os, const Resistencia * p) {
        return os << "Resistencia " << (ElementoPasivo *)p;
        }
};

class Bobina: public ElementoPasivo {
        friend class ElementoPasivo;
        Bobina(float v,unsigned i): ElementoPasivo(L,v,i) {}
public:
        friend ostream& operator<<(ostream& os, const Bobina * p) {
        return os << "Bobina " << (ElementoPasivo *)p;
        }

};

class Condensador: public ElementoPasivo {
        friend class ElementoPasivo;
        Condensador(float v,unsigned i): ElementoPasivo(C,v,i) {}
public:
        friend ostream& operator<<(ostream& os, const Condensador* p) {
        return os << "Condensador  " << (ElementoPasivo *)p;
        }

};
```

Examen de primer parcial (octubre 14)

```
void visualizar(ElementoPasivo *p){
          if(p->getTipo() == R) cout << (Resistencia *) p;
          else if(p->getTipo() == L) cout << (Bobina *) p;
          else cout << (Condensador *) p;
}
ElementoPasivo * ElementoPasivo::metodoFabricacion(TipoElemento t,float v,unsigned num)
{
          if(t==R) return new Resistencia(v,num);
          else if (t==L) return new Bobina(v,num);
          else return new Condensador(v,num);
}
class Cuadripolo {
          unsigned numR, numL, numC;
          std::vector<ElementoPasivo *> listaComponentes;
public:
          Cuadripolo():numR(0),numL(0),numC(0){}
          void addComponent(TipoElemento t, float v){
                    if(t==R)
                              listaComponentes.push_back(ElementoPasivo::metodoFabricacion(R,v,++numR));
                    else if(t==L)
                              listaComponentes.push_back(ElementoPasivo::metodoFabricacion(L,v,++numL));
                    else
                              listaComponentes.push_back(ElementoPasivo::metodoFabricacion(C,v,++numC));
          };
          ~Cuadripolo(){
                    for(unsigned i=0;i<listaComponentes.size();i++)
                              delete listaComponentes[i];
          }
          void list(){
          for_each(listaComponentes.begin(),listaComponentes.end(),visualizar);
          }
};
```

Examen de primer parcial (octubre 15)

Para el código de test adjunto, se pide:

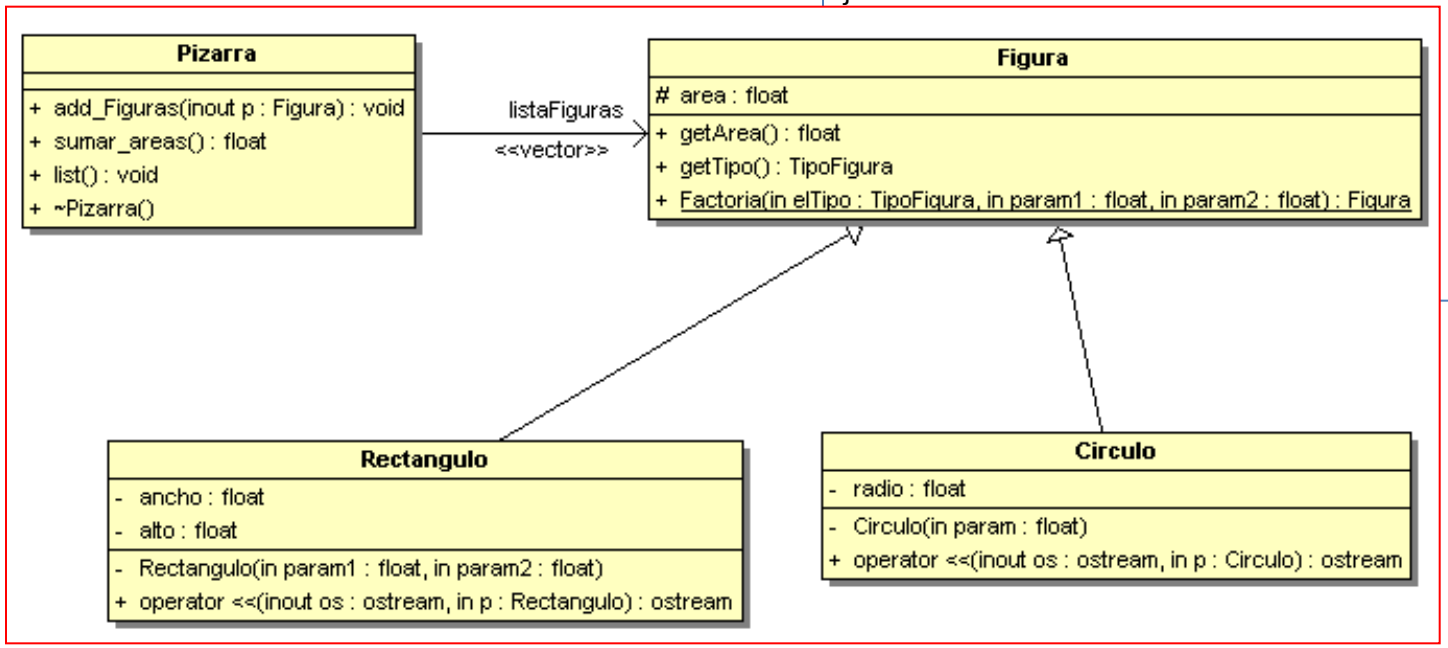
1. Diagrama de clases de diseño (2.5 puntos).
2. Implementación en C++ de la solución (7.5 puntos).

```
// Código de test
int main()
{
 Pizarra pizarra;
 pizarra.add_Figuras(Figura::Factoria(CIRCULO,1));
 pizarra.add_Figuras(Figura::Factoria(RECTANGULO,1,2));

 cout << "Lista de figuras:" << endl;
 pizarra.list();
 cout<<"Suma total area: "<<pizarra.sumar_areas()<<endl;

 return 0;
}
////////////////////////////////////
//Resultado en consola
////////////////////////////////////
Lista de figuras:
Circulo de radio 1
Rectangulo de ancho 1 y alto 2
Suma total area: 5.14152
```

Examen de primer parcial (octubre 15)

1. Diagrama de clases de diseño (2.5 puntos).

```
// Código de test
int main()
{
 Pizarra pizarra;
 pizarra.add_Figuras(Figura::Factoria(CIRCULO,1));
 pizarra.add_Figuras(Figura::Factoria(RECTANGULO,1,2));

 cout << "Lista de figuras:" << endl;
 pizarra.list();
 cout<<"Suma total area: "<<pizarra.sumar_areas()<<endl;

 return 0;
}
```



```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
#define PI 3.141516f
typedef enum{CIRCULO, RECTANGULO} TipoFigura;
class Figura {
protected:
 TipoFigura elTipo; float area;
public:
 float getArea() {return area;}
 TipoFigura getTipo() {return elTipo;}
 static Figura *Factoria(TipoFigura, float, float)
};
class Circulo : public Figura {
 friend class Figura; float radio;
 Circulo(float param):radio(param)
   {elTipo=CIRCULO;area=PI*radio*radio;}
public:
friend ostream&
perator<<(ostream& os, const Circulo * p){
 return os << "Circulo de radio " << p->radio <<
endl;
   }
};
class Rectangulo : public Figura {
friend class Figura; float ancho, alto;
Rectangulo(float param1, float param2)
          :ancho(param1),alto(param2)
  {elTipo=RECTANGULO;area=ancho*alto;}
public:
friend ostream&
operator<<(ostream& os, const Rectangulo * p) {
return os << "Rectangulo de ancho: " << p->ancho <
" y alto: "<< p->alto << endl;
}};
```

```
void visualizar(Figura *);
class Pizarra{
 vector<Figura *> listaFiguras;
public:
 void add_Figuras(Figura *p){
   listaFiguras.push_back(p);
  }
 float sumar_areas(){
  float area = 0;
   for(int i=0;i<listaFiguras.size();i++)
          area+=listaFiguras[i]->getArea();
          return (area);
    }
void list(){
for_each(listaFiguras.begin(),listaFiguras.end
(),
  visualizar);
}
~Pizarra(){
  for(int i=0;i<listaFiguras.size();i++)
          delete listaFiguras[i];
 }
};

Figura * Figura::Factoria(TipoFigura elTipo,
 float param1, float param2 = 0){
 if(elTipo == CIRCULO) return new
Circulo(param1);
 else return new Rectangulo(param1,param2);
}

void visualizar(Figura *p)
{
 if(p->getTipo() == CIRCULO) cout << (Circulo
*)p;
 else cout << (Rectangulo *)p;
}
```

Ejemplo de examen

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.

Ejemplo de examen

1. Jerarquía a dos niveles de las características principales.

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.

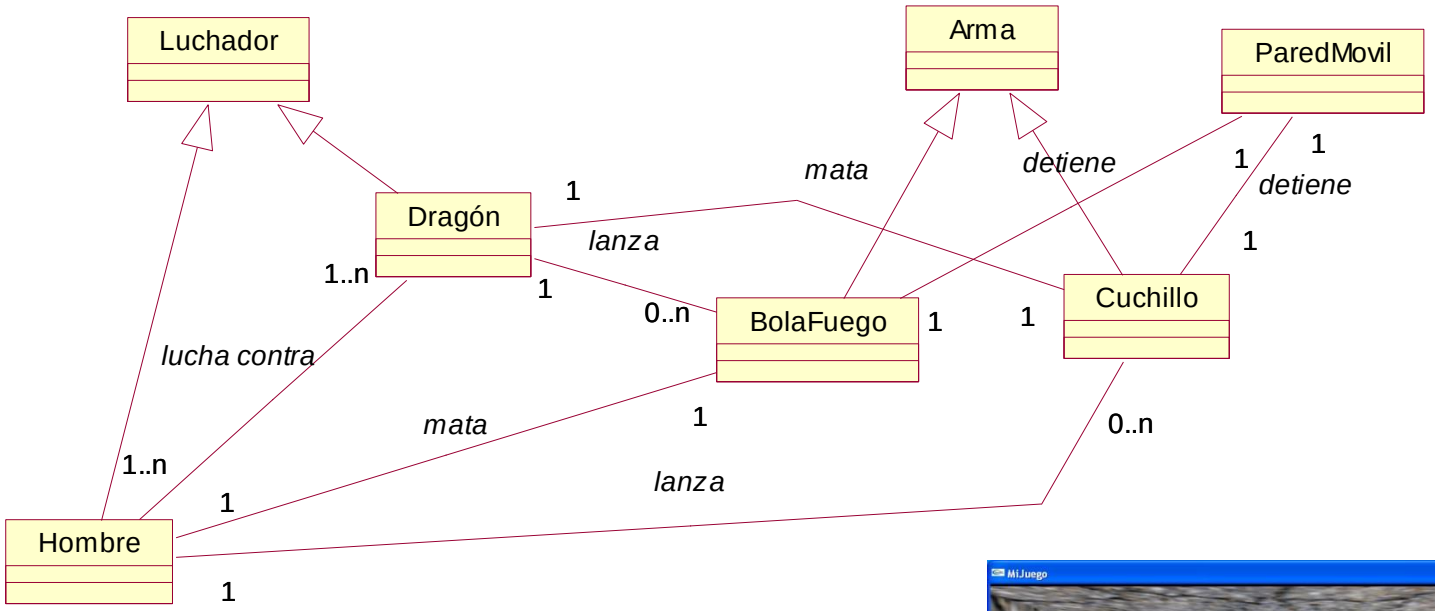
    1.4 Los dragones lanzan bolas de fuego que no pueden atravesar las paredes y que si tocan a un hombre lo mata.

    1.5 Los dragones desaparecen de la pantalla al morir.

    1.6 Los hombres desaparecen de la pantalla al morir.

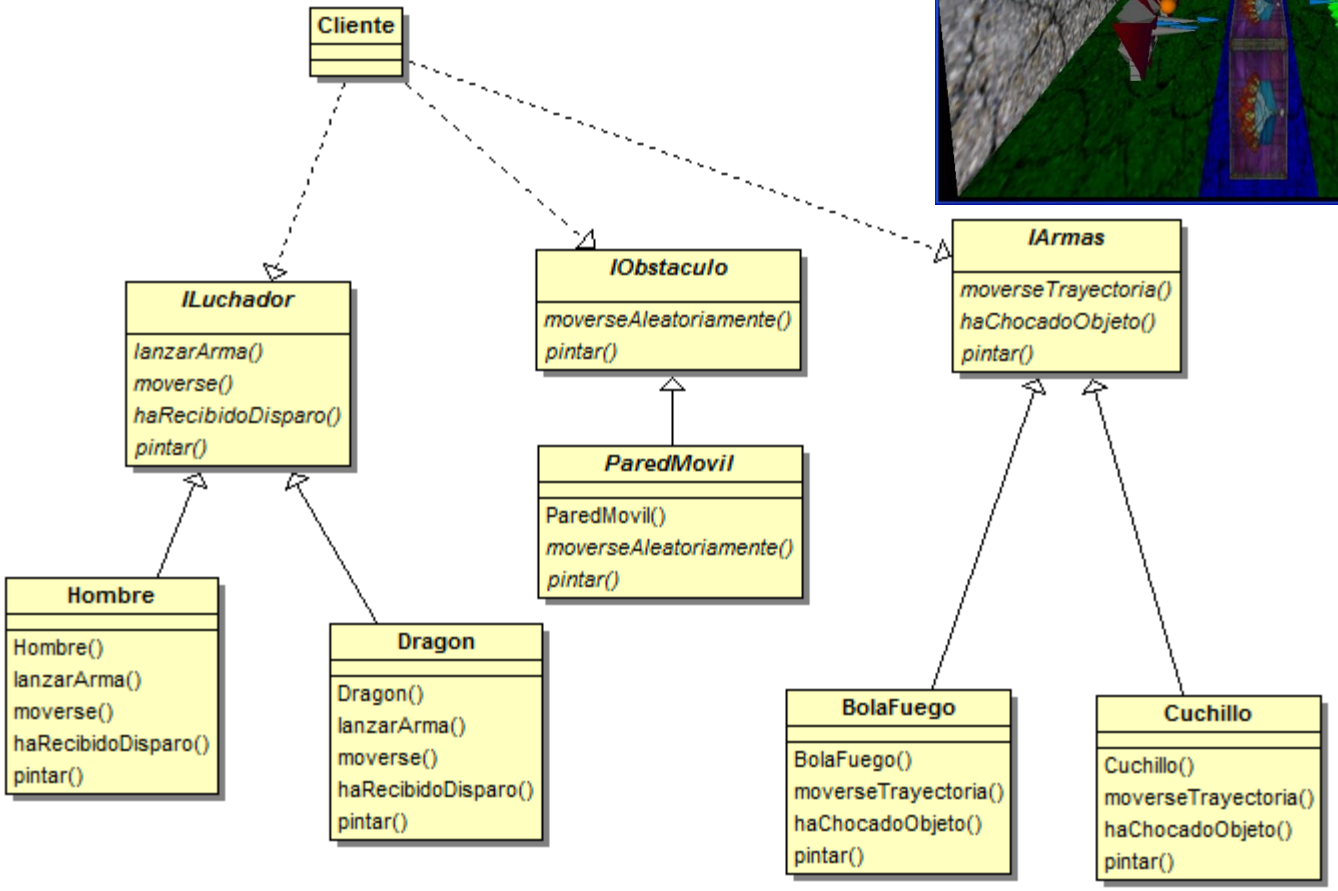
Ejemplo de examen

2. Modelo del dominio.



Ejemplo de examen

3. Diseño.



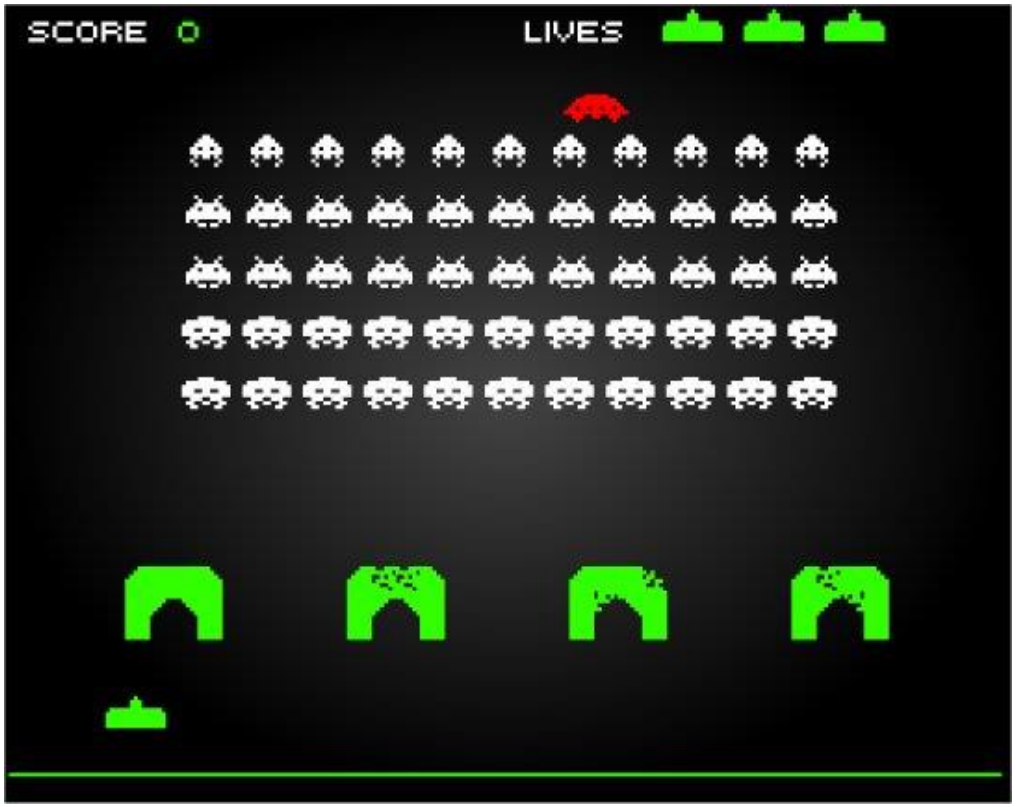
Ejemplo de examen

4. Implementación en C++

```
#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;
 Dragon();
public:
 virtual void lanzarArma();
 virtual void moverse();
 virtual bool haRecibidoDisparo();
 virtual void pintar();
};

#endif
```

```
#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:
virtual void moverseTrayectoria();
virtual bool haChocadoObjeto();
virtual void pintar();
};


#endif
```

Problemas de generalización



Práctica antigua de Informática Industrial

Realizar un programa para la gestión de las fichas de libros y revista en una biblioteca.

- La lista de características del sistema a dos niveles sería:

  1. Gestionar las fichas de una biblioteca

     1.a. Insertar, eliminar y listar las fichas.

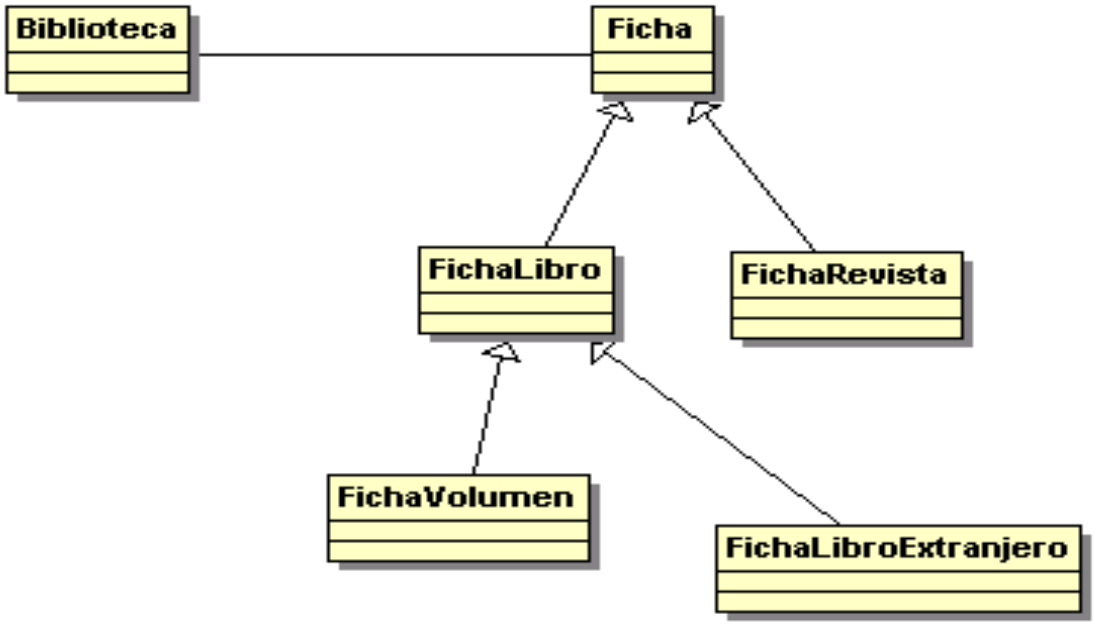
     1.b. Las fichas pueden ser de libros o de revistas. Los libros pueden estar constituidos por uno o más volúmenes. Los libros pueden ser en lengua extranjera.

     1.c. Las fichas de libros contendrán los campos de estante, posición, referencia, título, autor, editorial y volumen; mientras para las revistas deberán estar formados por el año y el número de la revista, junto con la referencia, el estante, la posición y el título.

- El glosario estaría constituido por las definiciones de: Ficha, Biblioteca, Libros, Revista, Referencia, Título,...

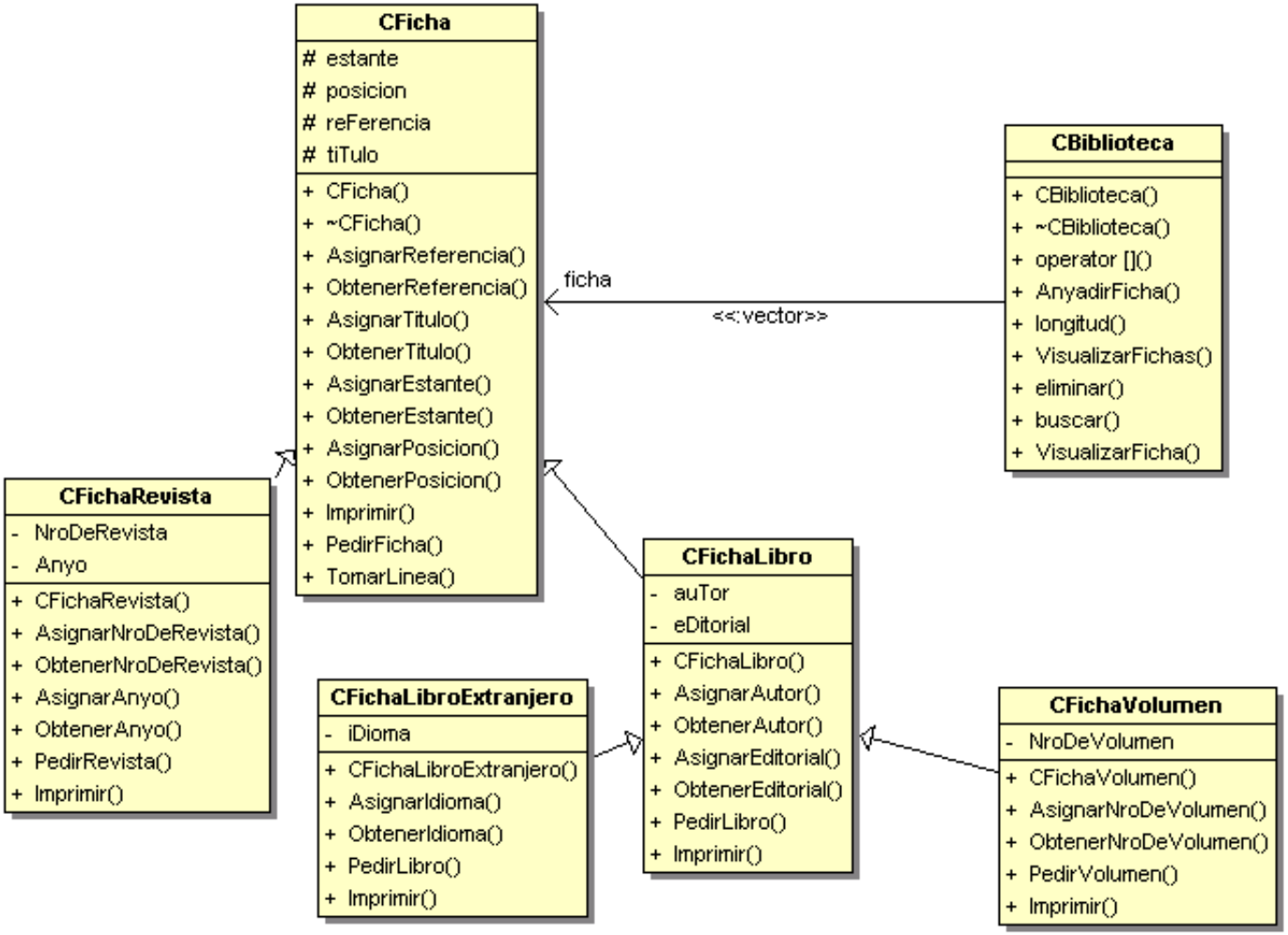
Práctica antigua

- El modelo del dominio podría ser del tipo:



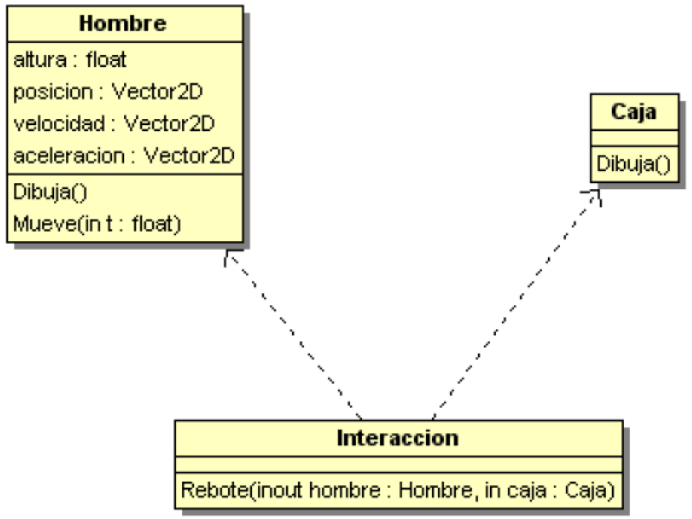
Práctica antigua

- Ingeniería inversa del código de la práctica



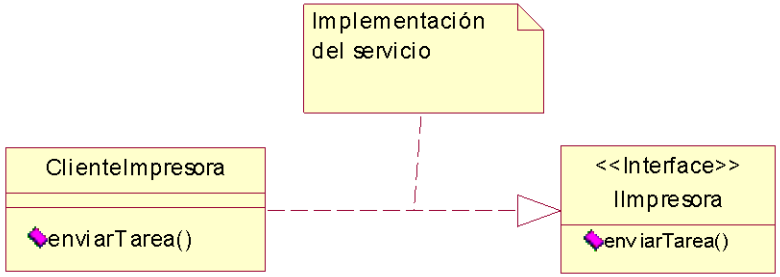
Dependencia

- Dependencia
  - Relación entre dos o más elementos
  - Un cambio en el servidor puede requerir cambios en el cliente.
  - No se añaden atributos (diferencia respecto a la asociación).
  - Cuando no es ni una asociación ni generalización.



Realización

- Realización
  - Conexión entre el cliente y la interfaz
  - Es un tipo de dependencia (se esconde la implementación)
  - Relación cliente-servidor
- Otros adornos UML:
  - Calificador
  - Asociaciones derivadas
  - Clases asociativas



Paquetes

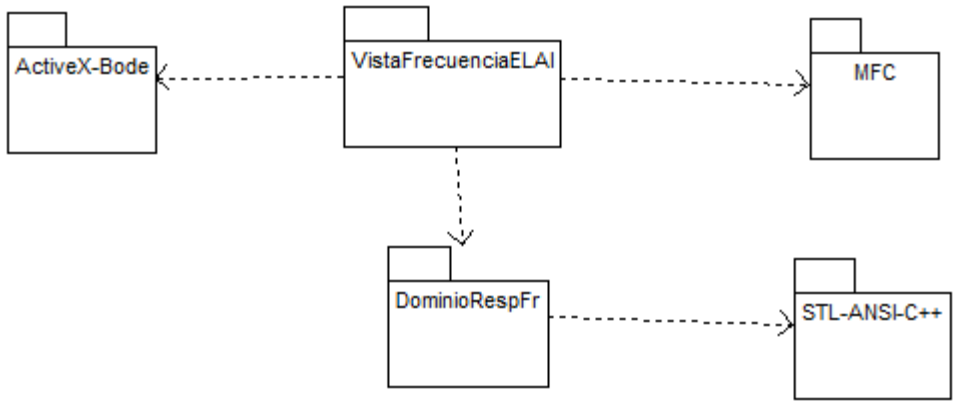
- Contenedor de: clases, paquetes, diagramas,..
- Propósito: la organización de cosas que tengan algo en común.
  - Agrupar clases altamente cohesivas
- Visibilidad y espacio de nombres
  - Fuera y dentro
  - Identificación UML: *paquete:nombre*

```
#include <iostream>

//Definición del espacio de nombres
using namespace std;

int main()
{
          cout<< "Hola Mundo\n";
}
```

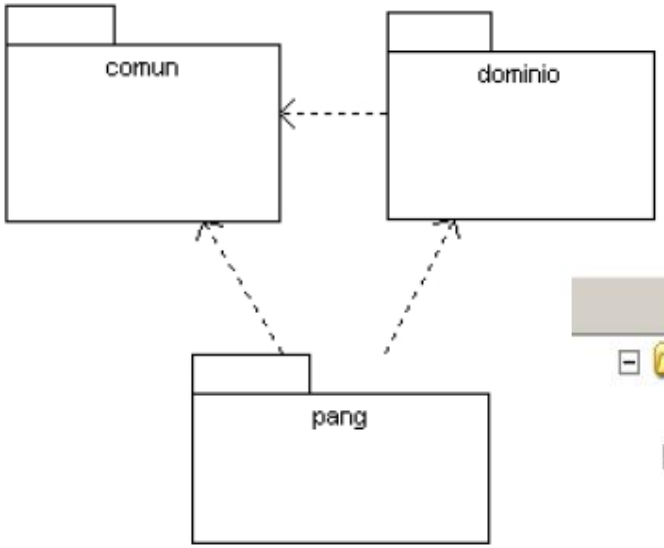
```
#include <iostream>

int main()
{
          //Uso del paquete estándar
          //y sintaxis de C++ entre
          //el paquete y el elemento
          std::cout<< "Hola Mundo\n";
}
```

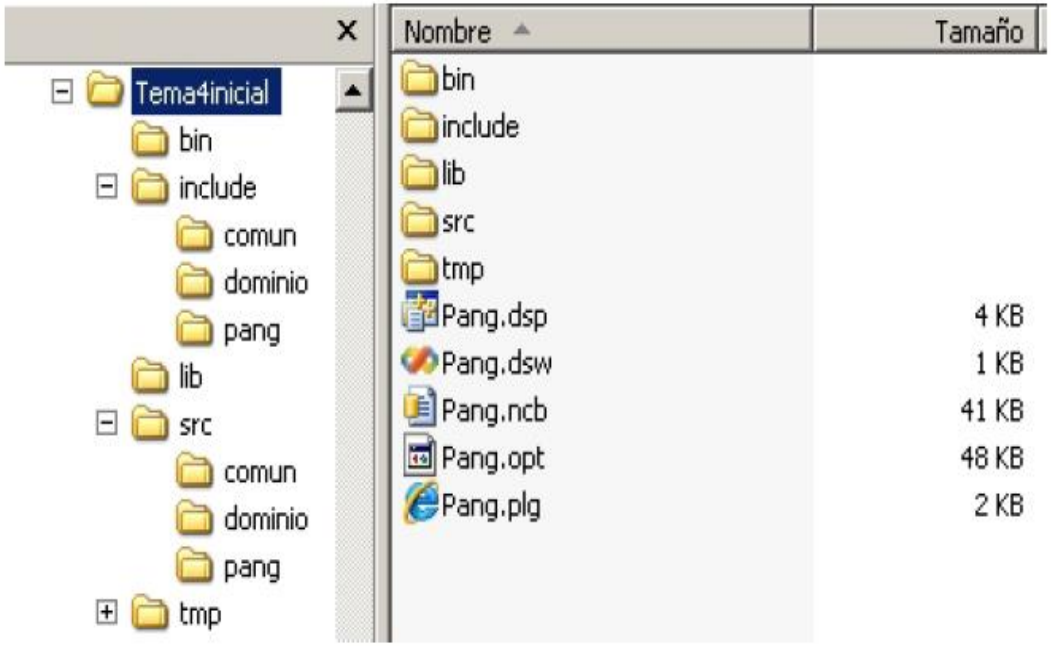
Paquetes

- Cómo dividir la aplicación en paquetes
  - Se encuentran en el mismo área de interés
  - Están juntos en una jerarquía de clases
  - Participan en los mismos caso de uso.
  - Están fuertemente asociados.
- Paquetes: Dominio, Común, …



Ejemplo del Pang





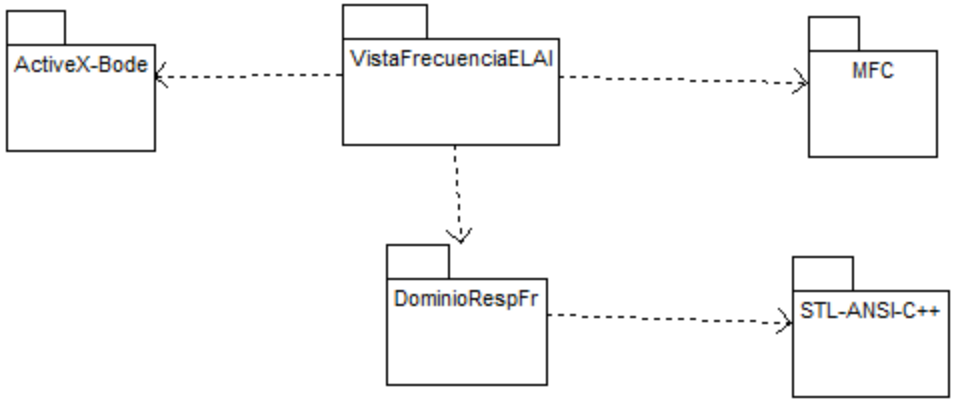
Paquetes

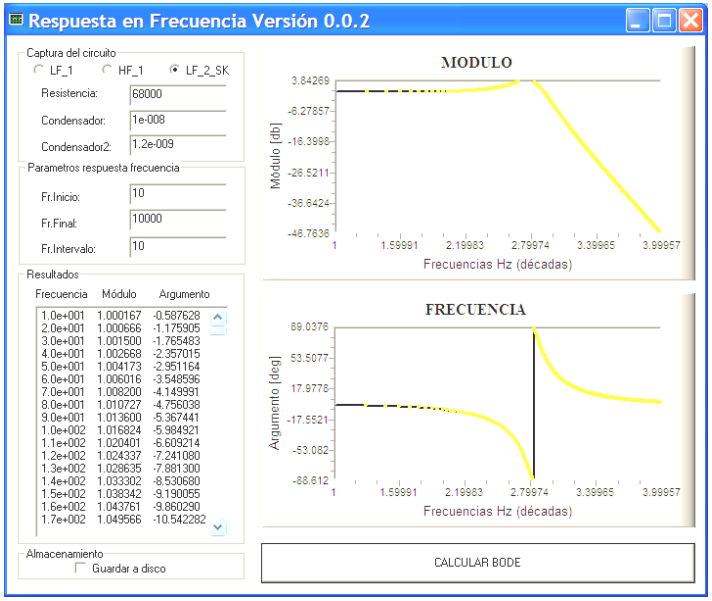
- Estabilidad del paquete
  - Reducir las dependencias de otros paquetes
    - Dependencias cíclicas: romperlas con interfaces.
    - Dependencias de paquetes estables
  - Vigilar los paquetes muy utilizados y en fase de elaboración. Cuidado con las revisiones.
  - Peligro: Paquetes con mucha responsables tiende a ser inestables
  - Cohesión interna:
  - Diseño: CR elevados
- Unidad de trabajo
- Vista de gestión : paquetes + dependencias

$$CR = \frac{Número\ de\ relaciones\ internas}{Número\ de\ tipos}$$

Ejemplo 4.10

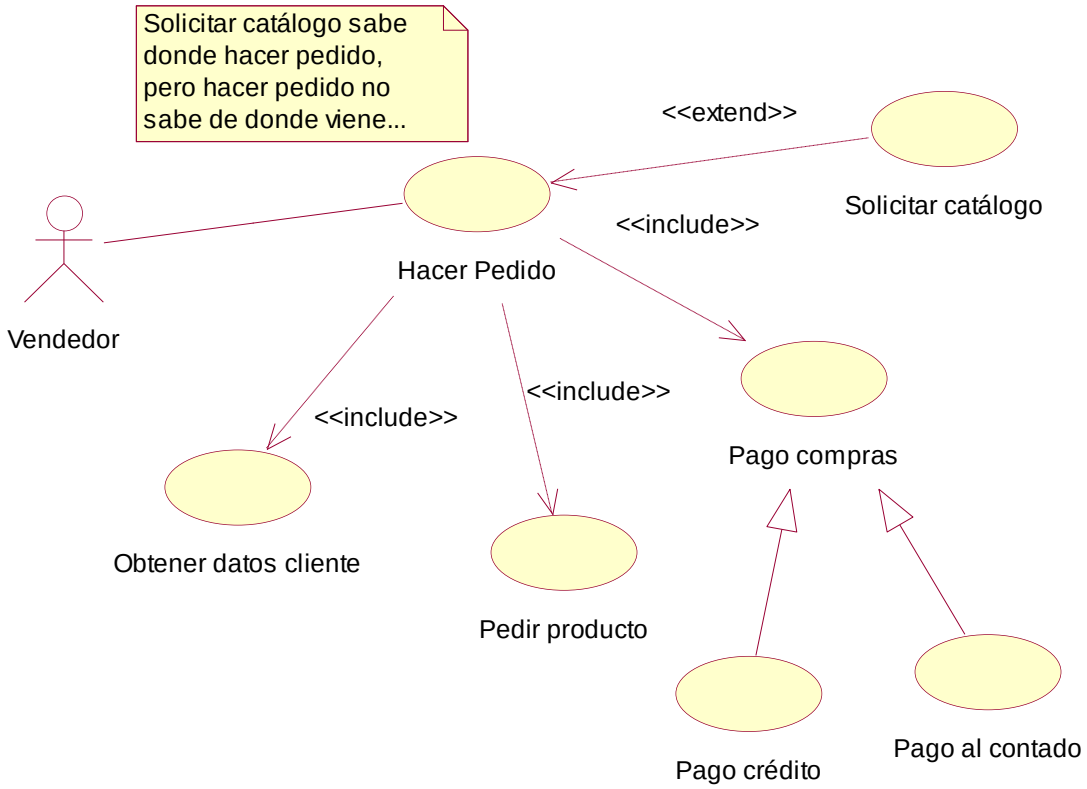
- Vista de gestión del modelo o diseño arquitectónico de la aplicación de respuesta en frecuencia.





Diagramas de casos de uso

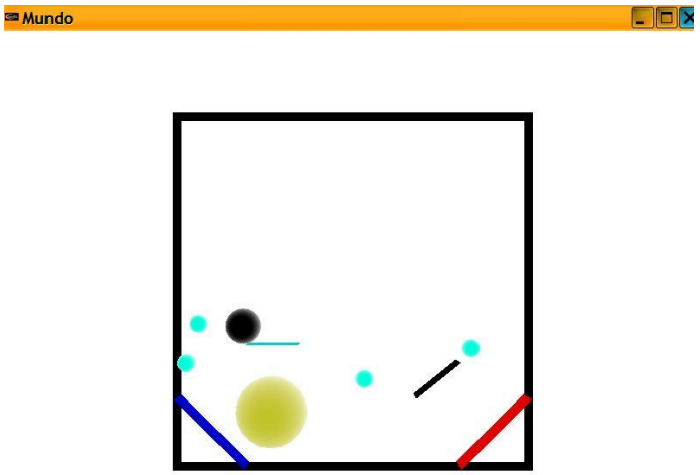
- Interacciones con el exterior.
- Sólo para ayudar a comprenderlos.
- Importante son los documentos.
- Relaciones
  - Factorización
  - Extensión
  - Generalización



Problema 4.3

Realizar una aplicación gráfica en cuyo mundo esté definida una CAJA que contiene un número indeterminado de ESFERAS. Se debe simular el rebote entre las propias ESFERAS y éstas con los SEGMENTOS de la CAJA.

1. Lista de características.
2. Modelo del dominio
3. Vista de gestión.
4. Diagrama de clases de diseño.



Problema 4.3

1. Lista de características.

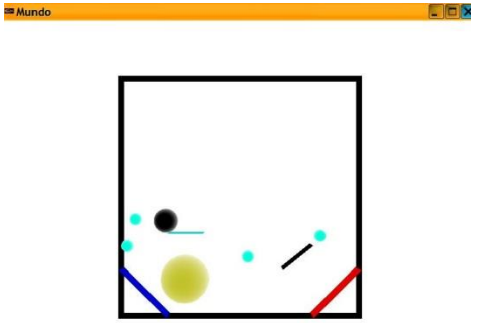
La lista de características del sistema a dos niveles sería:

1. Simulación de esferas que rebotan dentro de una caja

    1.a. Las esferas rebotan contra las paredes y barreras de la caja y con ellas mismas.

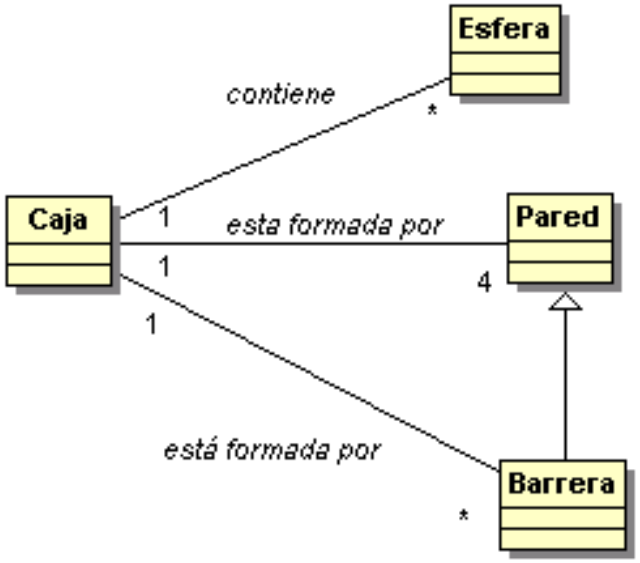
    1.b. La caja es cerrada y no se pueden escapar las esferas.

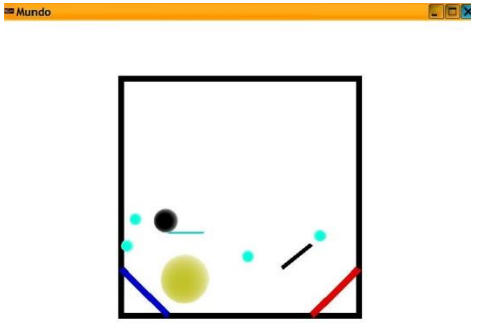
    1.c. La caja está formada por cuatro paredes y un número indeterminado de barreras.



Problema 4.3

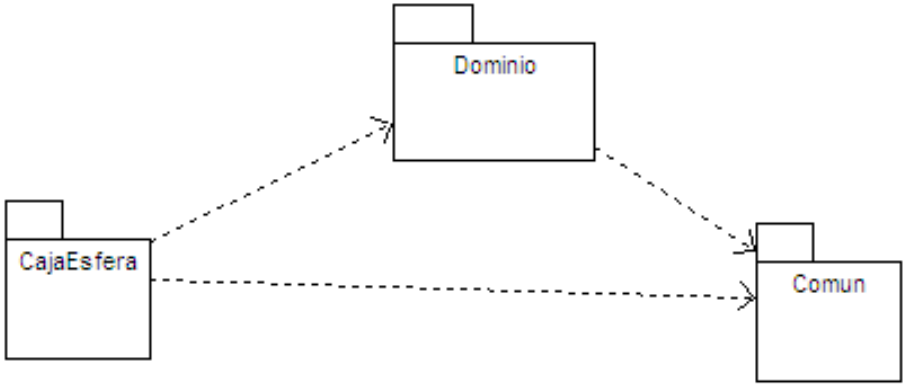
2. Modelo del dominio:

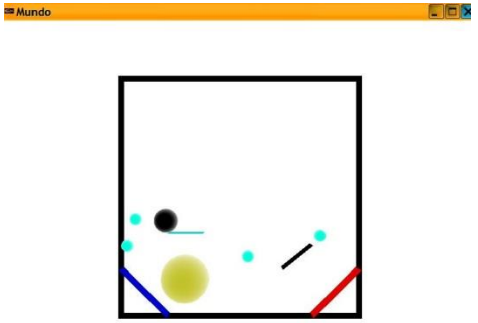




Problema 4.3

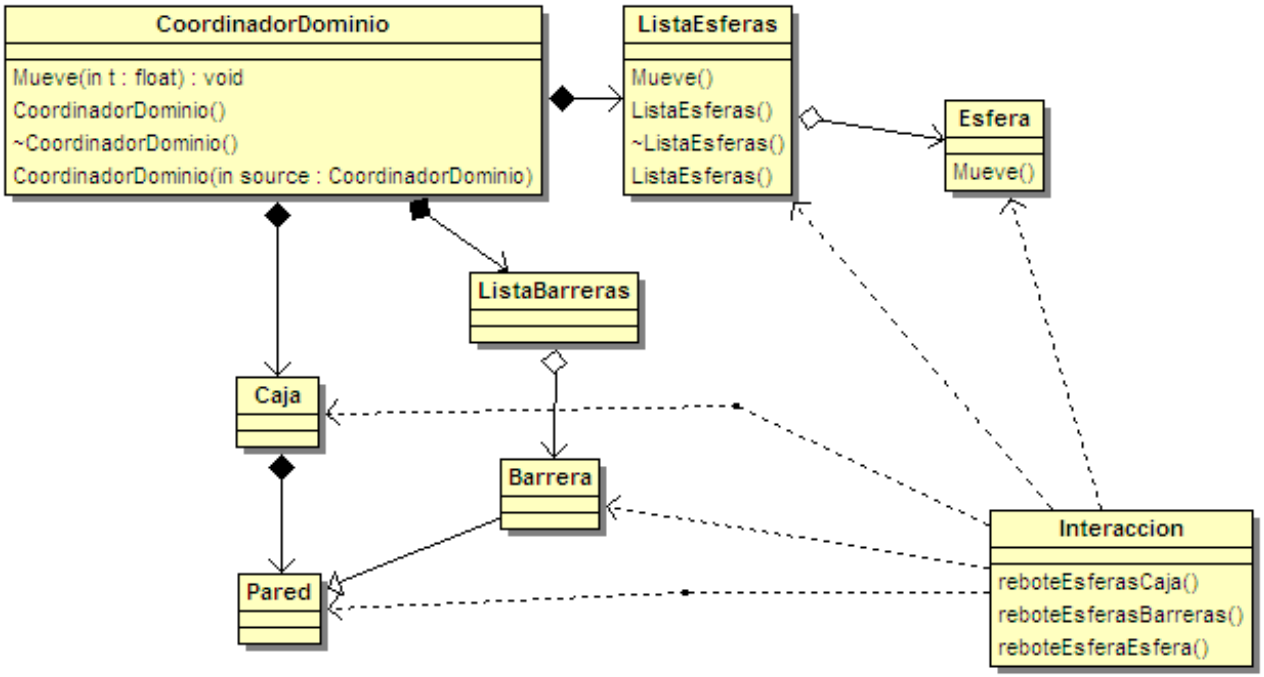
3.Vista de gestión

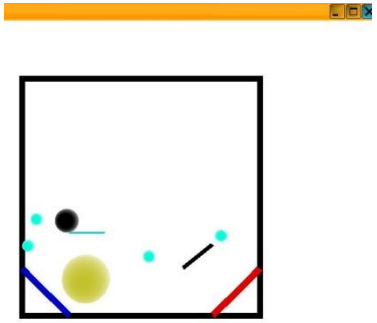




Problema 4.3

4. Diagrama de clase de diseño:





Cuestiones de UML estructural

1. Notación de UML sobre las clases.
2. Uso de las clases parametrizadas. Hágase un ejemplo sobre las pinturas de una galería de arte.
3. Defina un paquete sobre objetos geométricos, tales como cuadrado, rectángulo, triángulo, ... Defina un interface.
4. Defina una jerarquía de clases para los objetos geométricos del anterior pregunta. Presente las superclases y las subclases.
5. Cuando emplear una relación de asociación, agregación, composición, generalización y dependencia.
6. Cómo particionar la aplicación en paquetes.
7. Qué es la vista de gestión del modelo.
8. Relaciones en los diagramas de caso de uso.