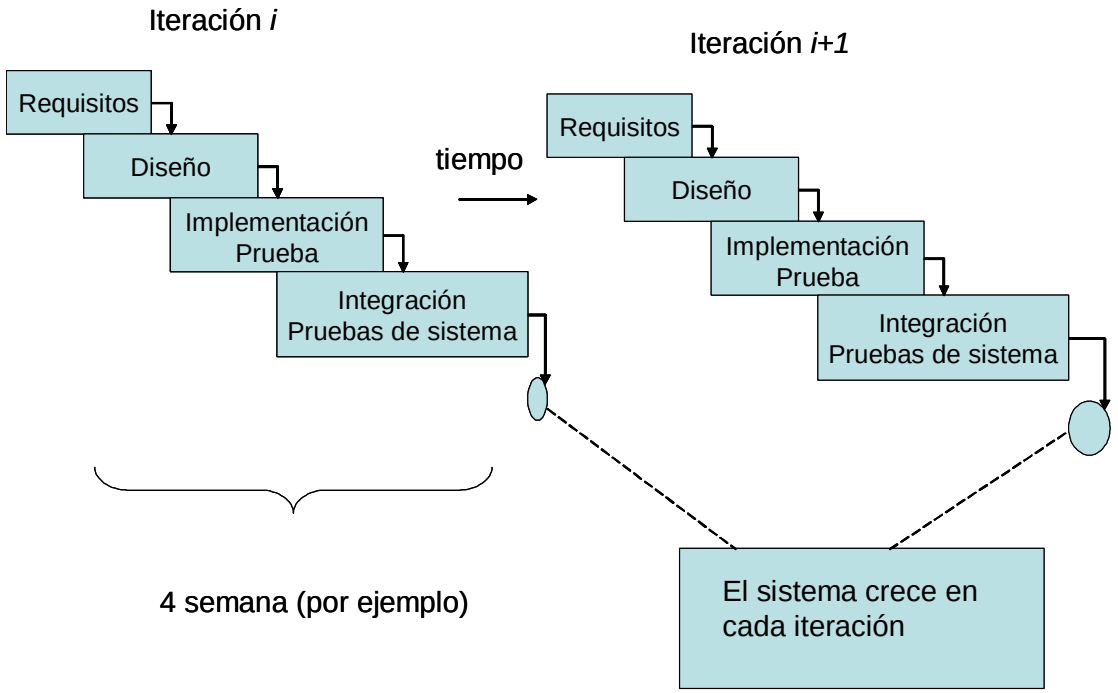
Capítulo VI: DOO (Parte I)

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

DOO

- AOO (describir) & DOO (solución)
- Implementar las especificaciones con eficiencia y fiabilidad
- Herramientas: Diagramas de interacción & DCD (en paralelo)
- DOO
  - Aplicar principios de asignación de responsabilidades
  - Aplicar patrones



Procedimiento DOO

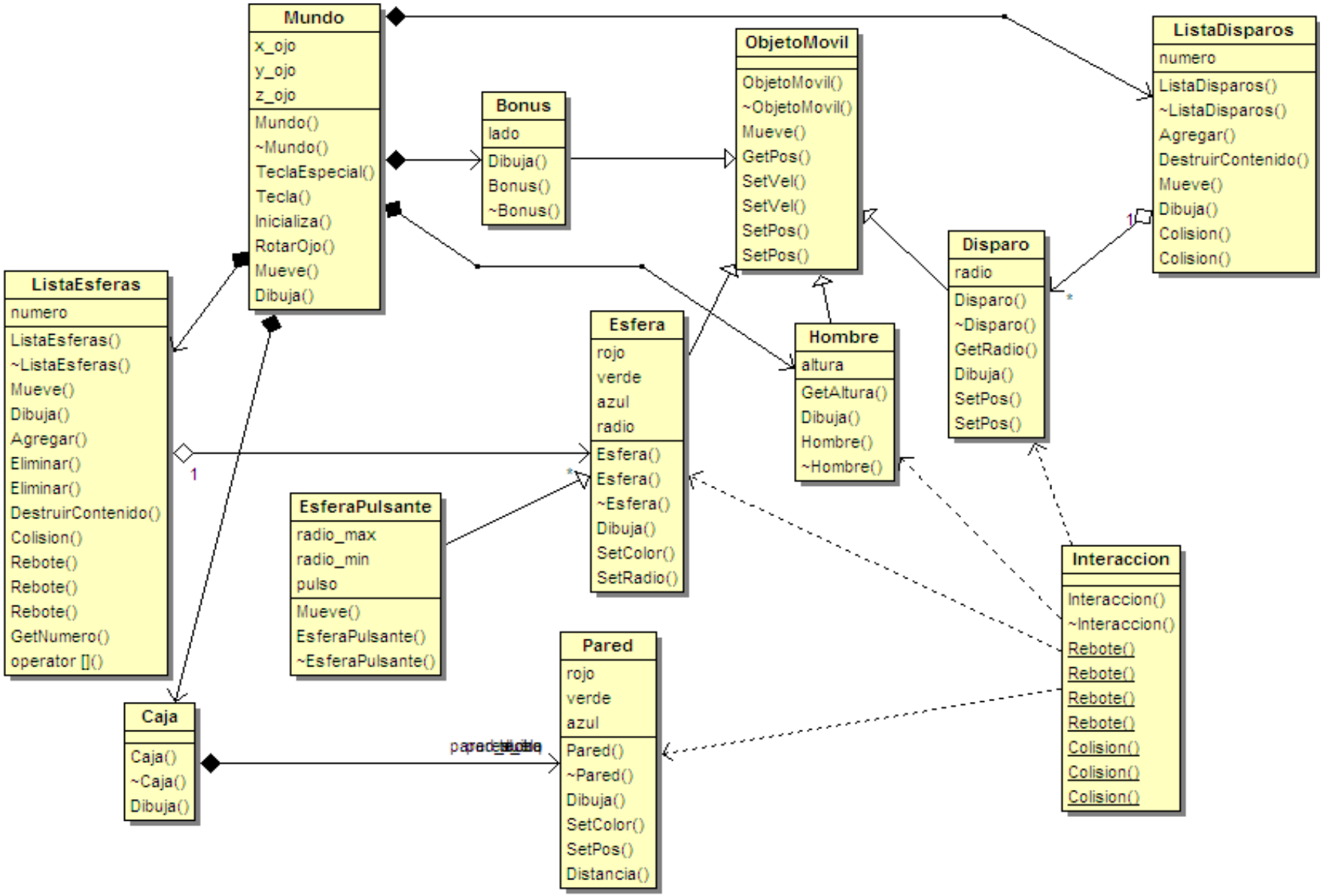
- Pasos
  1. Identificar las clases que participan en la solución del paquete
     1. Primera vez en el Modelo del dominio.
        1. De relaciones semánticas a asociaciones de *"necesito conocer"*.
     2. *Ingeniería inversa de la iteración anterior.*
  2. Asignar las responsabilidades con los diagramas de interacción.
     1. Create()
     2. Acceso (setX(), getX())
     3. Mensaje a multiobjetos se mandan al contenedor
  3. Dibujar DCD
     1. Indicar las visibilidades y las asociaciones

- Ahora se emplea toda la notación de UML

Visibilidad

- Para utilizar un servicio se requiere la *visibilidad* del objeto receptor (*el emisor manda un mensaje al receptor*)
  1. De atributo: la más empleada en POO <<association>>
  2. De parámetro: el receptor es pasado como un argumento <<parameter>>
  3. Local: se declara el receptor en algún servicio del emisor <<local>>
  4. Global: el receptor es declarado global. Uso del patrón GoF *Singleton*. <<global>>
- Navegabilidad flujo de los datos con visibilidad
- De atributo se refleja con la asociación, el resto se representa con una relación de dependencia

Ejemplo de visibilidades



Del DOO hacia la Programación

- Modelo de Implementación: entrada DCD y los diagramas de interacción.
- Trabajo no trivial. Menos productivo en caso de no utilizar el UP (guía de principio a fin).
- Transformación
  - Definiciones de las clases desde los DCDs
  - Definiciones de los métodos desde los diagramas de interacción.
- XP (e*Xtreme Programming*)
  - Código de test, luego código de producción
  - De la menos a la más acoplada

Ejemplo: Respuesta en frecuencia (1)

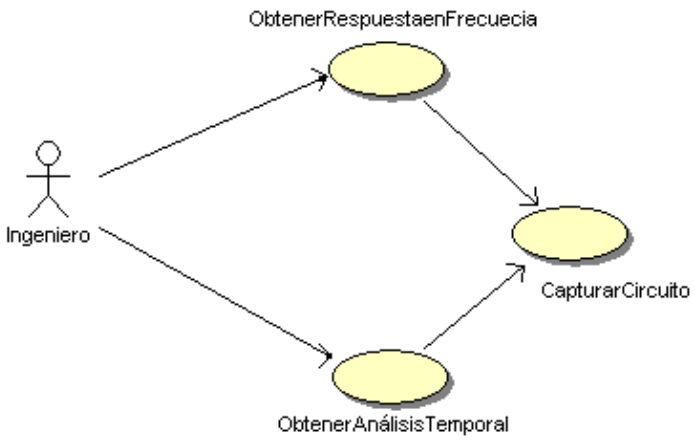
Ejemplo 2.2

Diseñar una aplicación que entregue la respuesta en frecuencia de los filtros analógicos lineales. Una vez capturado el circuito eléctrico, se pasará a determinar cual es la función de transferencia en el dominio de la frecuencia y se presentará la respuesta en frecuencia en diagrama de Bode.

Ejemplo 2.4

Determinar las características para la aplicación de Respuesta en Frecuencia enunciada en el ejemplo 2.2. Utilícese un esquema de dos niveles.

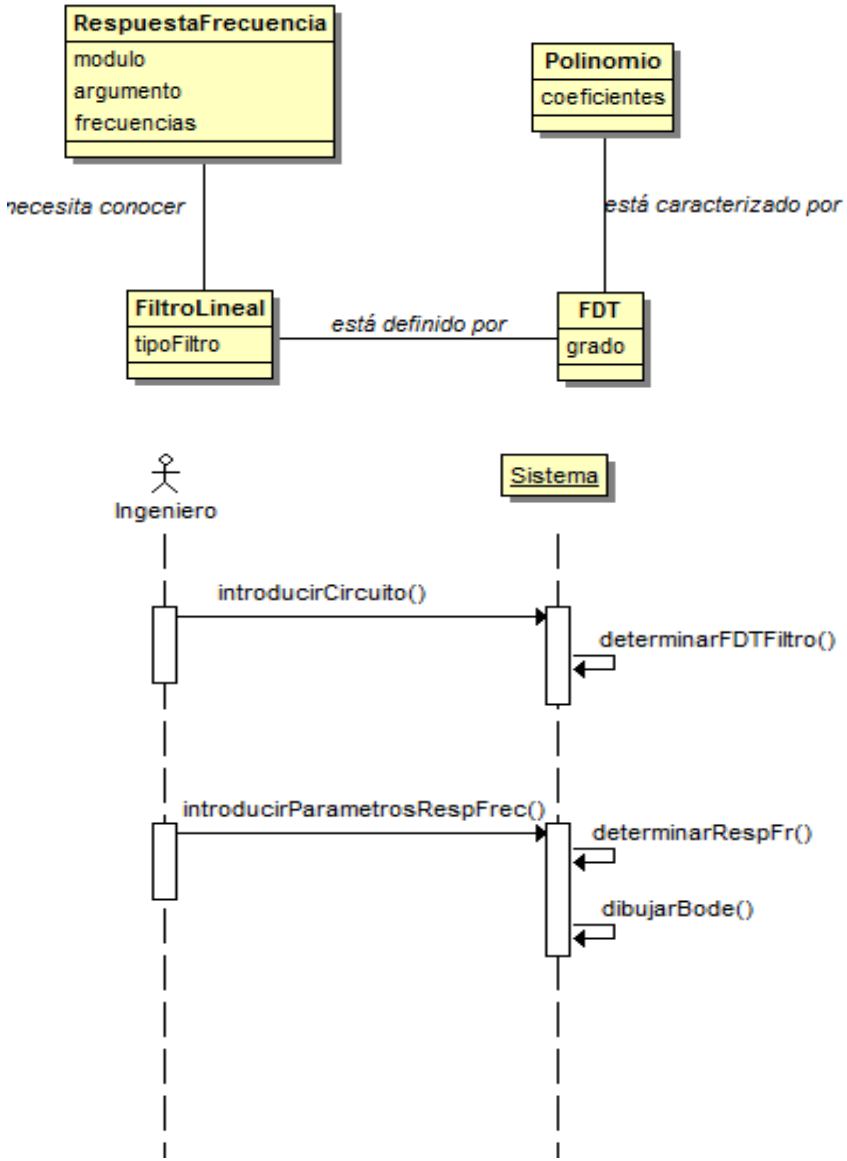
- Captura del circuito
  - Interacción con el usuario para determinar el circuito analógico
  - Determinar la FDT del circuito lineal.
- Análisis en frecuencia
  - Parámetros de la respuesta en frecuencia (Rango de frecuencia, intervalo en el cálculo, lineal o en décadas)
  - Presentación gráfica del diagrama de Bode.



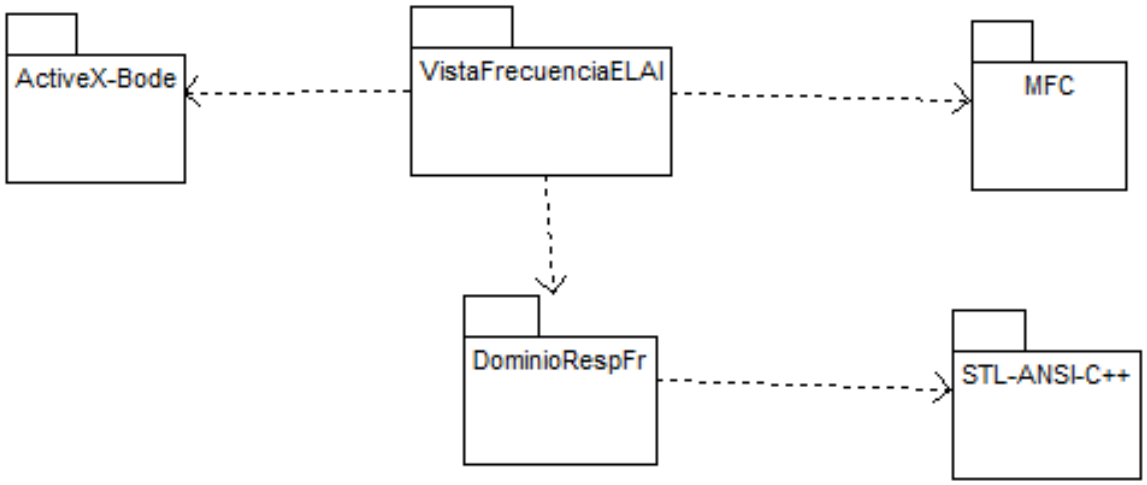
Ejemplo: Respuesta en frecuencia (2)

Actores:	Ingeniero electrónico
Descripción:	El ingeniero elige el tipo de filtro lineal e introduce los valores de los componentes analógicos, luego elige los parámetros de la respuesta en frecuencia y la aplicación le devolverá la respuesta en frecuencia en un diagrama de Bode
Precondiciones:	El ingeniero conoce y sabe todos los parámetros del filtro lineal y de la respuesta en frecuencia.
Poscondiciones:	Se presentará en diagrama de Bode la respuesta en frecuencia del filtro capturado
Curso normal:	1.0.El ingeniero introduce el circuito eléctrico (mirar caso de uso incluido de capturar el circuito) 2.0. La aplicación pide el rango de frecuencias y el intervalo aplicado en la determinación de la respuesta en frecuencia. Además se le solicitará si desea una presentación en decibelios o lineal. 3.0. La aplicación calcula la FDT del circuito y determina la respuesta en frecuencia. 4.0. Los resultados son presentados en un diagrama de Bode (módulo/argumento).
Curso alternativo:	1.1 El circuito es capturado desde un fichero de descripción de componentes electrónicos tipo * CIR
Excepciones:	3.0.E.1 Hay algún error de desbordamiento o de división por cero en el cálculo de la respuesta en frecuencia. Se le notificará del error matemático al usuario
Inclusiones:	Captura del circuito
Prioridad:	Máxima. Núcleo del sistema.
Frecuencia de uso:	Podría ser casi continuo.
Reglas de negocio:	
Requerimientos especiales:	
Suposiciones de partida:	
Notas y documentos:	Documento de adquisición de los circuitos

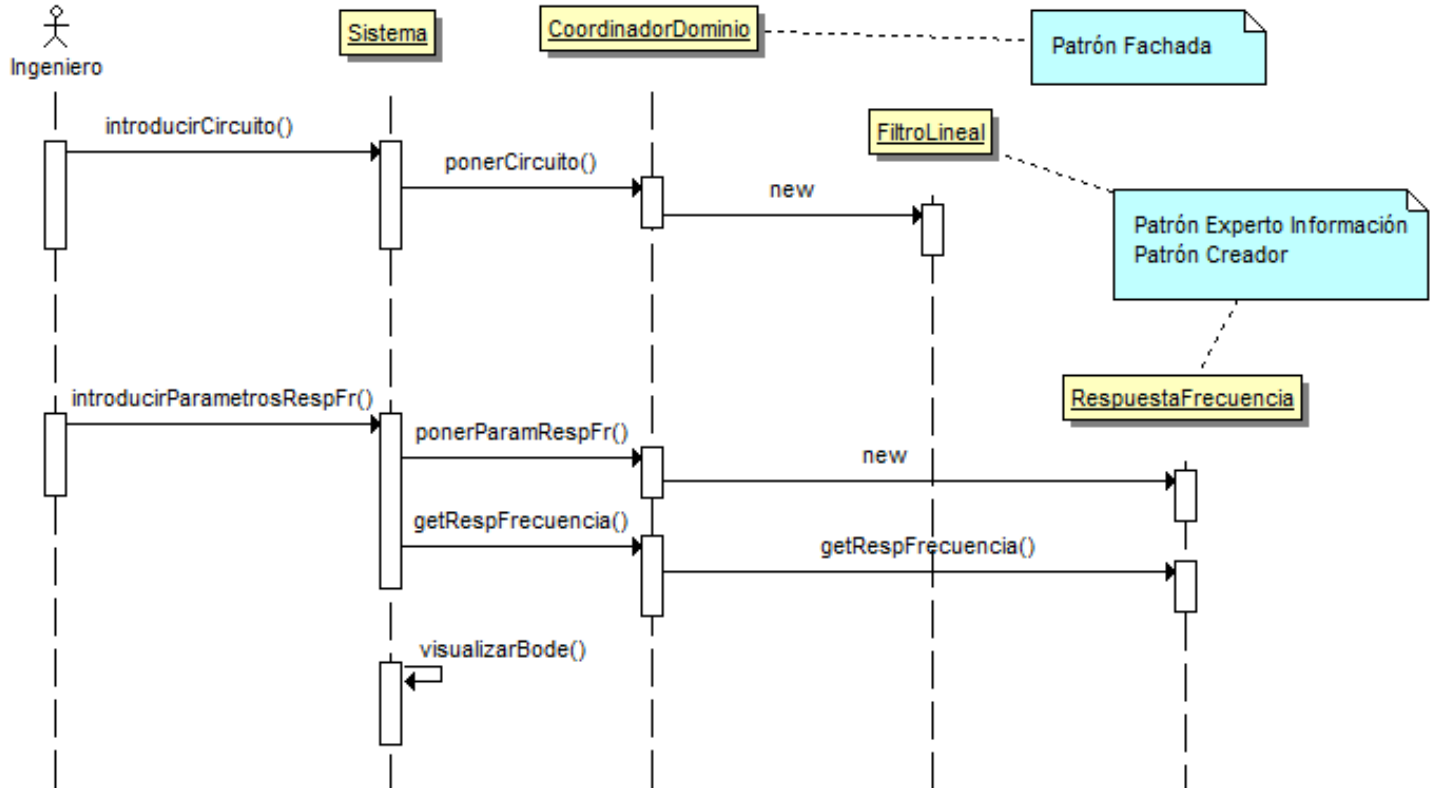
Ejemplo: Respuesta en frecuencia (3)



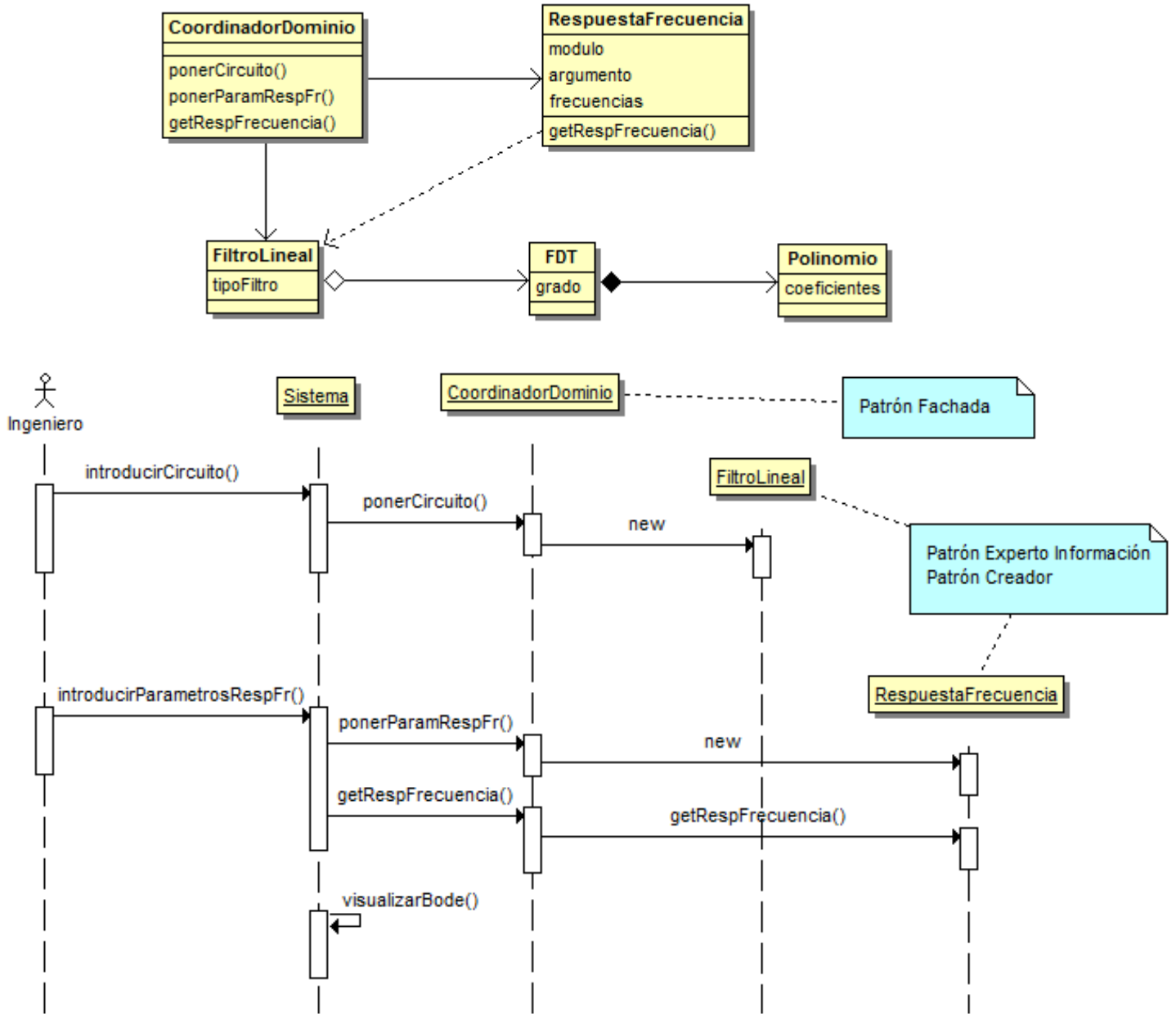
Ejemplo: Respuesta en frecuencia (4)



Ejemplo: Respuesta en frecuencia (5)



Ejemplo: Respuesta en frecuencia (6)



Ejemplo: Respuesta en frecuencia (7) (Cód. Test)

```
void VistaFrecuenciaELAI::introducirCircuito(void){
cout << "Elegir entre:\n1.Filtro paso bajo primer orden.\n2.Filtro paso alto";
          int eleccion;       cin >> eleccion;
          elTipo = eleccion == 1 ? LF_1 : HF_1;
          cout << "\nValor de la resistencia: ";  cin       >> resistencia;
          cout << "\nValor del condensador: ";    cin >> condensador;
          elCoordinador.ponerCircuito(elTipo,resistencia,condensador);
}
void VistaFrecuenciaELAI::introducirParametrosRespFr(void){
          cout << "\nCual es la frecuencia inicial [Hz]: "; cin >> frecInicial;
          cout << "\nCual es la frecuencia final [Hz]: ";   cin >> frecFinal;
          cout << "\nCual es el intervalo empleado para el cálculo [Hz]: ";
          cin >> frecIntervalo;
          elCoordinador.ponerParamResFr(frecInicial,frecFinal,frecIntervalo);
          //Visualizar los resultados
          std::vector<double> elVectorModulo;
          elCoordinador.getModuloRespFr(elVectorModulo);
          for (unsigned i =0; i<elVectorModulo.size(); i++)
                    std::cout<<elVectorModulo[i]<<std::endl;
          cout << "Pulsar cualquier tecla para finalizar";
}
void main(void){
          VistaFrecuenciaELAI laVista;
          laVista.introducirCircuito();
          laVista.introducirParametrosRespFr();
}
```

Ejemplo: Respuesta en frecuencia (8) (Cód. Prod.)

```
#include "FiltroLineal.h"
#include "RespuestaFrecuencia.h"

class CoordinadorFrecELAI
{
            FiltroLineal *pFiltro;
            RespuestaFrecuencia *pRespFr;

public:
            int ponerCircuito(tipoFiltro ,float , float );
            int ponerParamResFr(float,float,float);
            int getModuloRespFr(std::vector<double> &);
            ~CoordinadorFrecELAI()
            {if(pFiltro) delete pFiltro; if(pRespFr) delete pRespFr;}

};
```

```
#include "../../include/Dominio/CoordinadorFrecELAI.h"
int CoordinadorFrecELAI::ponerCircuito(tipoFiltro elTipo,float resistencia, float condensador){
            pFiltro = new FiltroLineal(elTipo,resistencia,condensador);
            return(0);
}
int CoordinadorFrecELAI::ponerParamResFr(float frInicio,float frFinal,float frIntervalo){
            if (pFiltro == NULL) return (-1);
            pRespFr = new RespuestaFrecuencia(frInicio,frFinal,frIntervalo,pFiltro);
            return(0);
}
int CoordinadorFrecELAI::getModuloRespFr(std::vector<double> &elVectorModulo){
            if (pRespFr == NULL) return (-1);
            pRespFr->getModuloRespFr(elVectorModulo);
            if(pFiltro) delete pFiltro;
            if(pRespFr) delete pRespFr;
            return (0);
}
```

Ejemplo: Respuesta en frecuencia (9) (Cód. Prod.)

```
#ifndef _POLINOMIO_INC_
#define _POLINOMIO_INC_
#include <vector>

class Polinomio
{
  std::vector<double> coeficientes;
public:
  Polinomio(){}
  Polinomio(unsigned grado, double *pCoef){
    for (unsigned i=0;i<=grado; i++)
            coeficientes.push_back(*(pCoef+i));
   }
   double getCoeficiente(unsigned n)
   {return( coeficientes[n]));}
};
#endif /*Polinomio.h*/
```

```
#ifndef _FDT_INC_
#define _FDT_INC_
#include "Polinomio.h"
class FDT
{
 unsigned grado;
 Polinomio numerador;
 Polinomio denominador;
public:
 FDT(unsigned n, double *pNum, double *pDen):
    grado(n),numerador(n,pNum),denominador(n,pDen){}
 unsigned getGrado(void){return grado;}
 double getCoefNum(unsigned n)
    {return n<=grado ? numerador.getCoeficiente(n) : 0;}
 double getCoefDen(unsigned n)
    {return n<=grado ? denominador.getCoeficiente(n) : 0;}
};
#endif /*FDT.h*/
```

Ejemplo: Respuesta en frecuencia (10) (Cód. Prod.)

```
#include "FDT.h"
typedef enum{LF_1,HF_1} tipoFiltro;

class FiltroLineal
{
            tipoFiltro elTipo;
            FDT *pFDT;
public:
            FiltroLineal(tipoFiltro, float, float);
            unsigned getGradoFiltro(void){return pFDT->getGrado();}
            double getCoefNum(unsigned n)
            {return pFDT != NULL ? pFDT->getCoefNum(n) : 0;}
            double getCoefDen(unsigned n)
            {return pFDT != NULL ? pFDT->getCoefDen(n) : 0;}
            ~FiltroLineal(){if(pFDT) delete pFDT;}
};
```

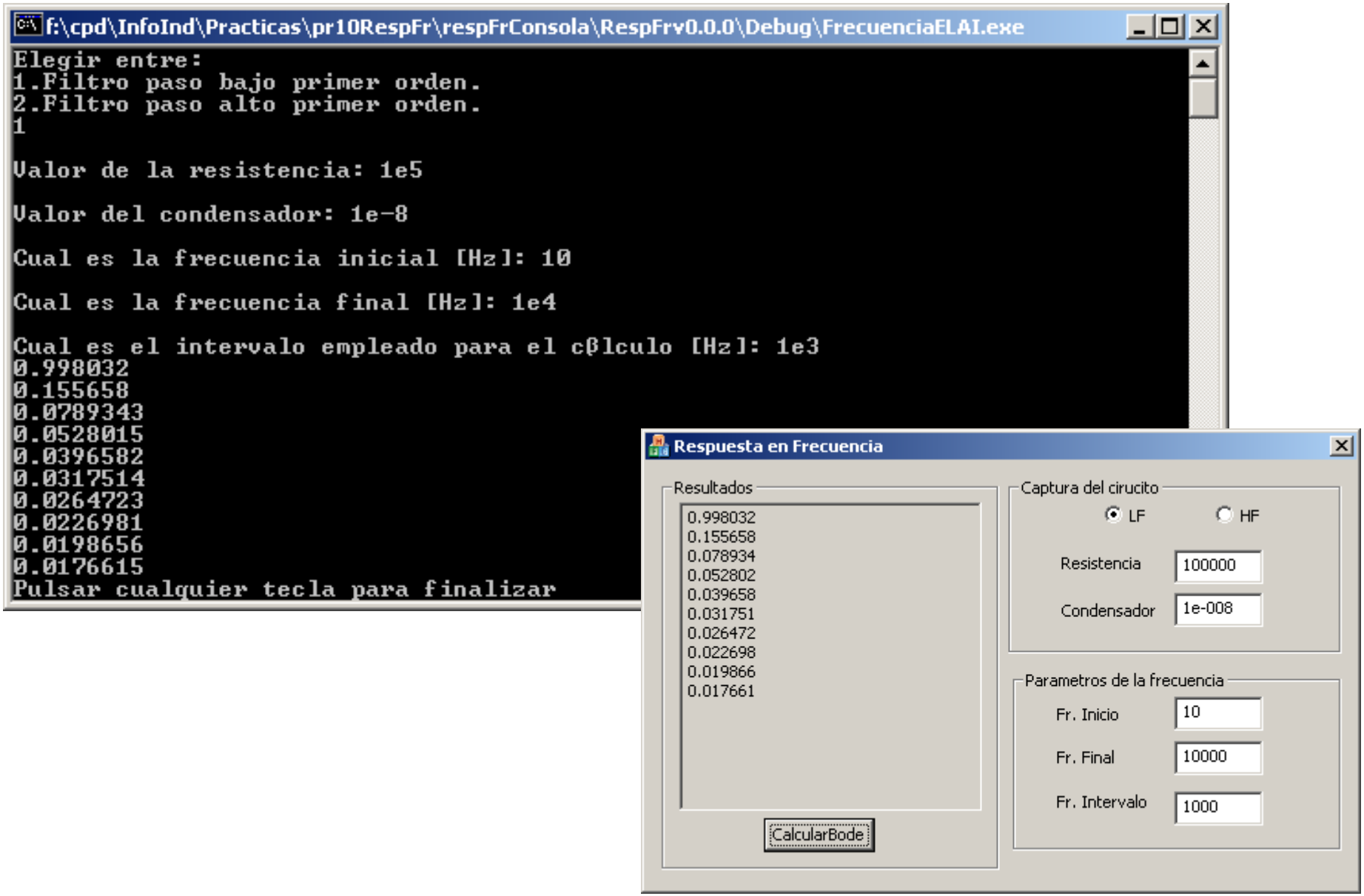
```
#include "../../include/Dominio/FiltroLineal.h"
FiltroLineal::FiltroLineal(tipoFiltro tipo, float resistencia, float condensador)
{
      elTipo = tipo;
      double numerador[2]; double denominador[2];
      if (elTipo == LF_1) {
                  numerador[0]=1; numerador[1]=0;
      }
      else{
                  numerador[0]=0; numerador[1]=resistencia*condensador;
      }
      denominador[0]= 1;denominador[1]=resistencia*condensador;
pFDT = new FDT(1,numerador,denominador);
}
```

Ejemplo: Respuesta en frecuencia (11) (Cód. Prod.)

```
#include <vector>
#include "FiltroLineal.h"
class RespuestaFrecuencia
{
            float freInicio, freFinal, freIntervalo;
            std::vector<double> modulo;
            std::vector<double> argumento;
            double calcularModulo(float,FiltroLineal *);
            double calcularArgumento(float,FiltroLineal *);
public:
            RespuestaFrecuencia(float,float,float,FiltroLineal *);
            float getFrInicio(void){return freInicio;}
            float getFrFinal(void){return freFinal;}
            float getFrIntervalo(void){return freIntervalo;}
            void getModuloRespFr(std::vector<double> &elVectorModulo)
                        {elVectorModulo = modulo;}
};
```

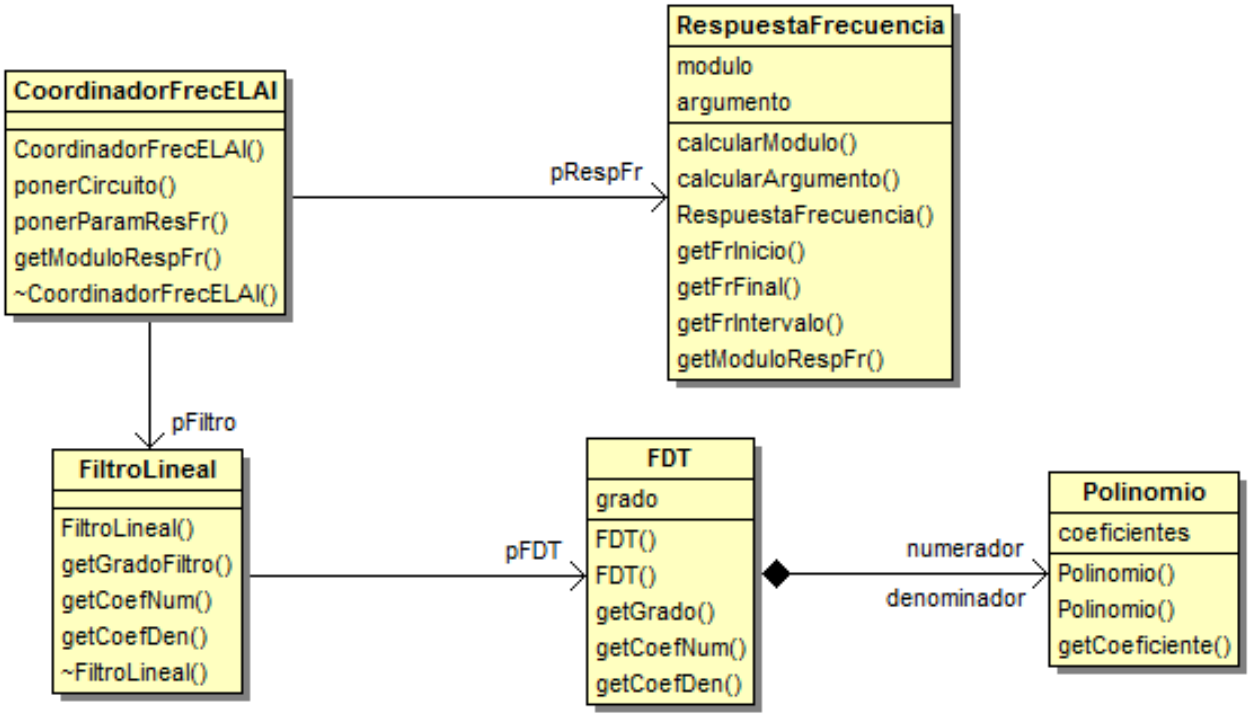
```
#define PRUEBA_BODE
#ifdef PRUEBA_BODE
#include <iostream>
#endif
RespuestaFrecuencia::RespuestaFrecuencia(float frInicio, float frFin,float
      frIntervalo,FiltroLineal *pFiltro){
      for (float f=frInicio; f< frFin; f+=frIntervalo){
                  modulo.push_back(this->calcularModulo(f,pFiltro));
                  //argumento.push_back(this->calcularArgumento(f,pFiltro));
      }
#ifdef PRUEBA_BODE
      this->iteradorModulo = modulo.begin();
      for (unsigned i =0; i<modulo.size(); i++)
                  std::cout<<*(iteradorModulo+i)<<std::endl;
#endif
}
```

Ejemplo: Respuesta en frecuencia (12)



Ejemplo: Respuesta en frecuencia (13)

- Ingeniería inversa



Diseño con responsabilidad

- Responsabilidad de un objeto
  - A) La información que maneja (conocer)
  - B) Las cosas que debe de hacer (procesar)
- Las responsabilidades se asigna en el DOO.
- Las responsabilidades se implementan mediante servicios.
- Patrón: problema-solución + nombre + aplicable a diferentes contextos.
  - GRASP (*General Responsibility Assignment Software Patterns):*
    - Experto en Información, Creador, Alta Cohesión, Bajo Acoplamiento, Controlador, Polimorfismo, Indirección, Fabricación Pura y Variaciones Protegidas.
  - GoF (*Gangs of Four*) :
    - Adaptador, Factoría, Singleton, Estrategia, Composición y Observador.

Experto en Información (GRASP)

Problema: ¿Cuál es el principio general para asignar responsabilidades?

Solución: Asignar la responsabilidad al que tenga la información.

- Realizar tabla de responsabilidades:
  - Clase, Información y Responsabilidades,
- Indica lo que hacen los objetos con su información.

Ejemplo 6.3

Realizar una tabla de responsabilidad sobre la aplicación *RespuestaFrecuencia*

Clase de diseño	Información	Responsabilidad
Filtro	Tiene la FDT del filtro y el tipo de filtro	Definir matemáticamente la estructura del filtro
RespuestaFrecuencia	Los parámetros de frecuencia	Aplicar los algoritmos para calcular el Bode

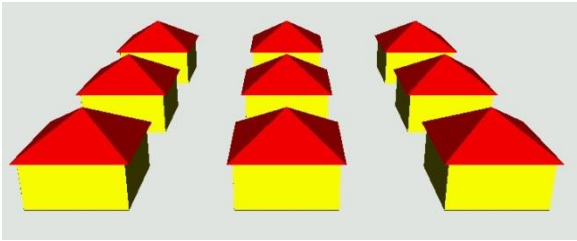
Experto en Información (GRASP)

- Beneficios:
  - Mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas.
  - Se distribuye el trabajo entre clases, haciéndolas más cohesivas y ligeras, lo que conlleva a que sean más fáciles de entender y de mantener.
- Inconveniente:
  - Acoplamiento entre paquetes
    - Mantener separado las distintas lógicas

Ejemplo de Experto en Información: 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)



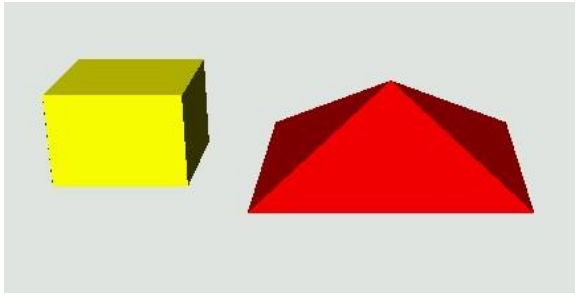
El ancho de las casas es de 1 unidad, su altura de 0.5 y la separación entre casas es de 3 unidades.

Ejemplo: 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;
};
```

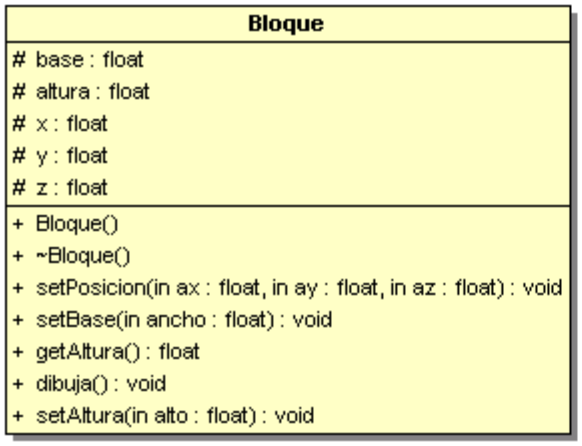
```
class Bloque
{
public:
 float getAltura();
 void setAltura(float a);
 void setBase(float b);
 void setPos(float px, float py,
             float pz);
 void dibuja();

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

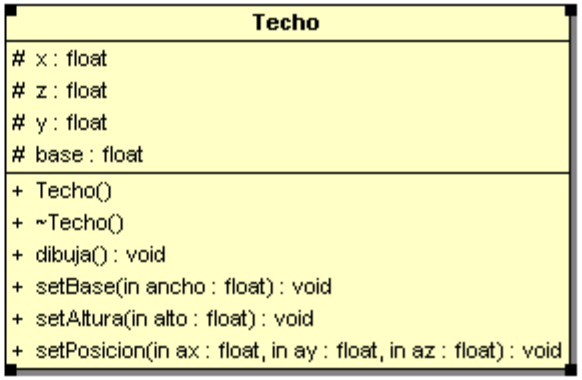
```
void main()
{
 Techo t;
 Bloque b;
 t.setPos(0,0,0);
 t.setBase(2);
 b.setPos(2,0,0);
 b.setAltura(0.7);
 b.setBase(1);
 t.dibuja();
 b.dibuja();
}
```



Ejemplo: Urbanización (1)

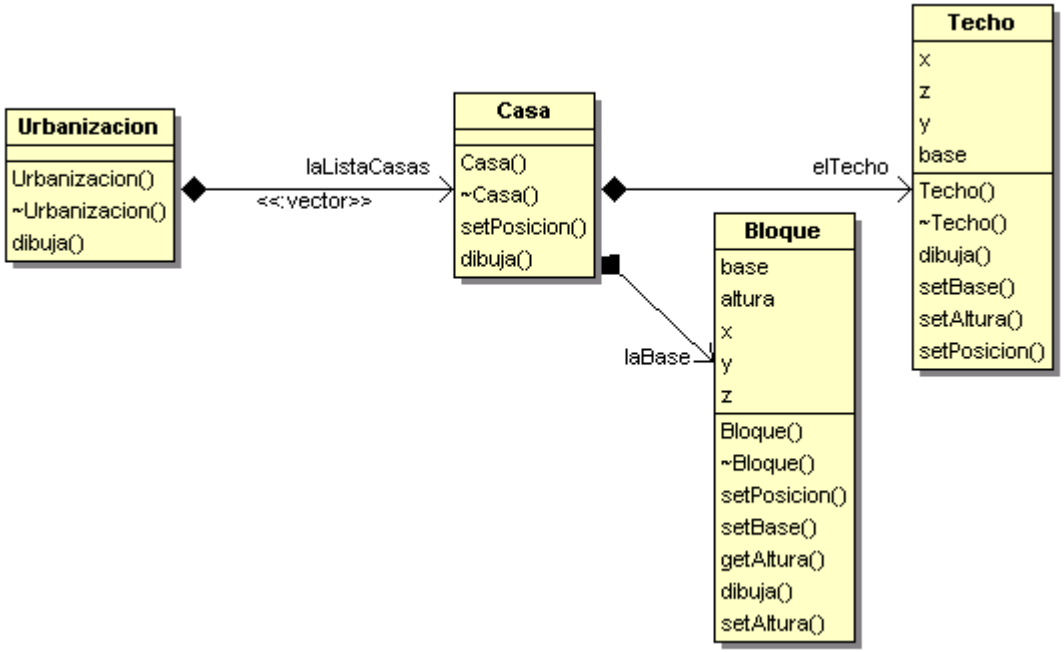
1. Representación UML de las clases Techo y Bloque.





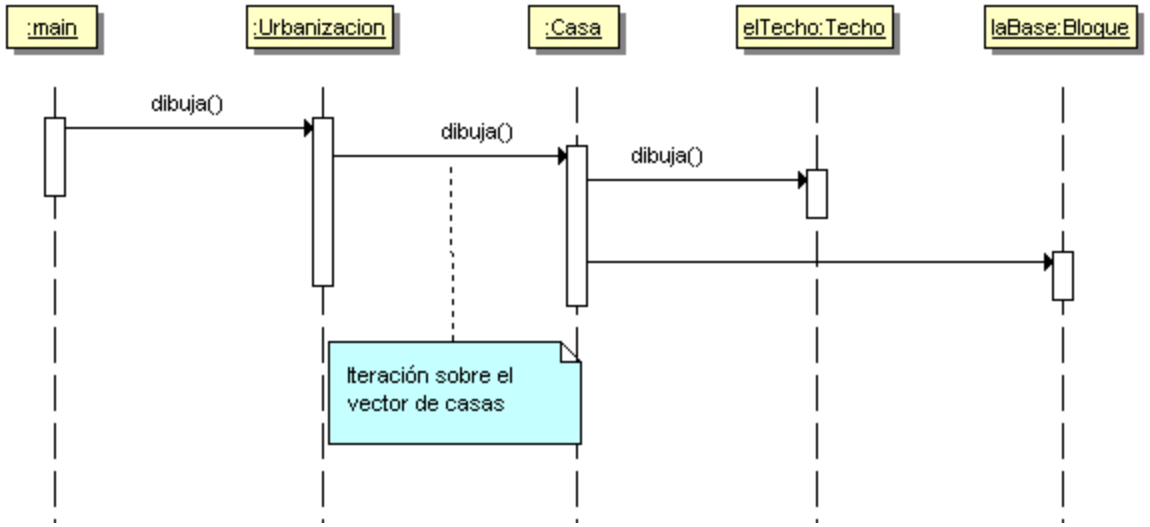
Ejemplo: Urbanización (2)

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



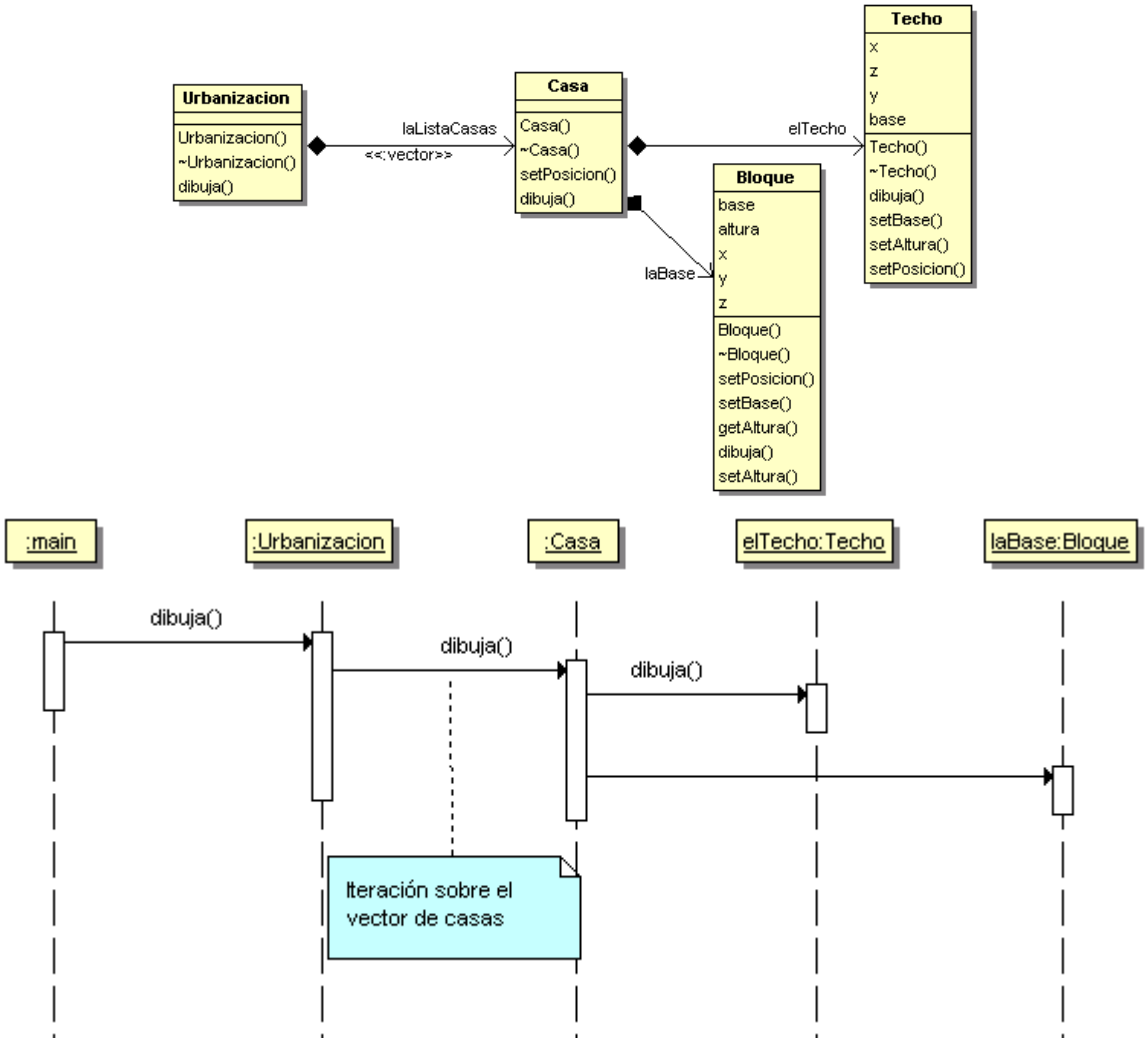
Ejemplo: Urbanización (3)

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



Ejemplo: Urbanización (4)

4. Implementación en C++



Ejemplo: Urbanización (5)

4. Implementación en C++

   Código de test

```
int main()
{
        Urbanizacion elPoblado(4,5);
        elPoblado.dibuja();
        return 0;
}
```

Ejemplo: Urbanización (6)

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();
}
```

Ejemplo: Urbanización (7)

4. Implementación en C++

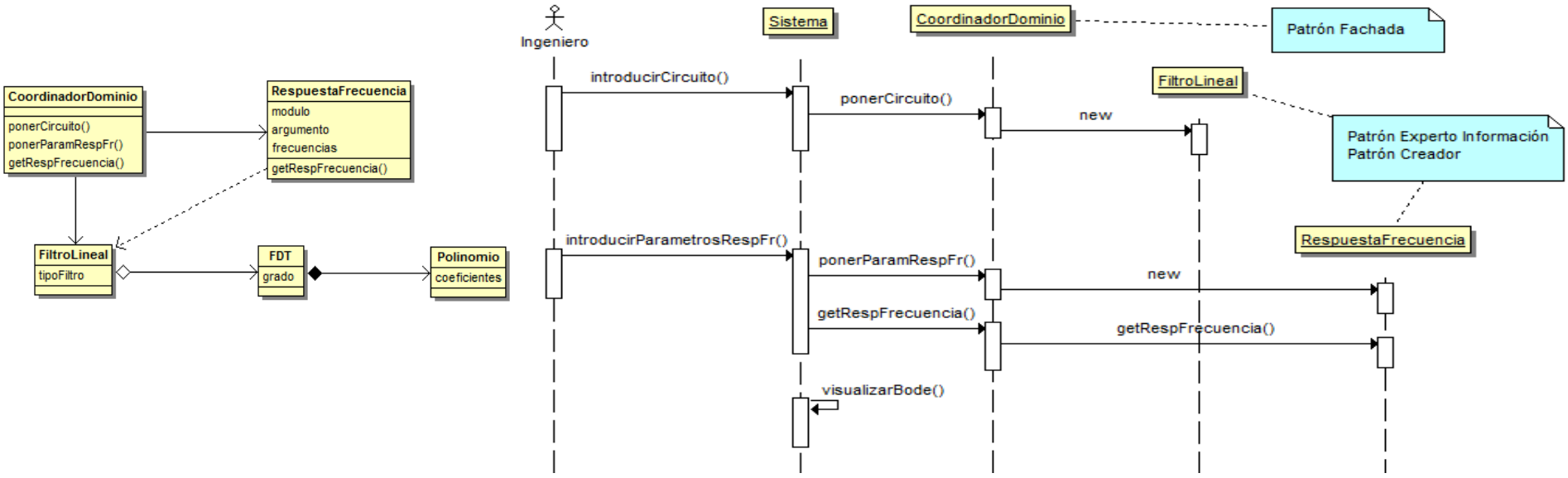
```
#include <vector>
#include "Casa.h"

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();
}
```

Creador (GRASP)

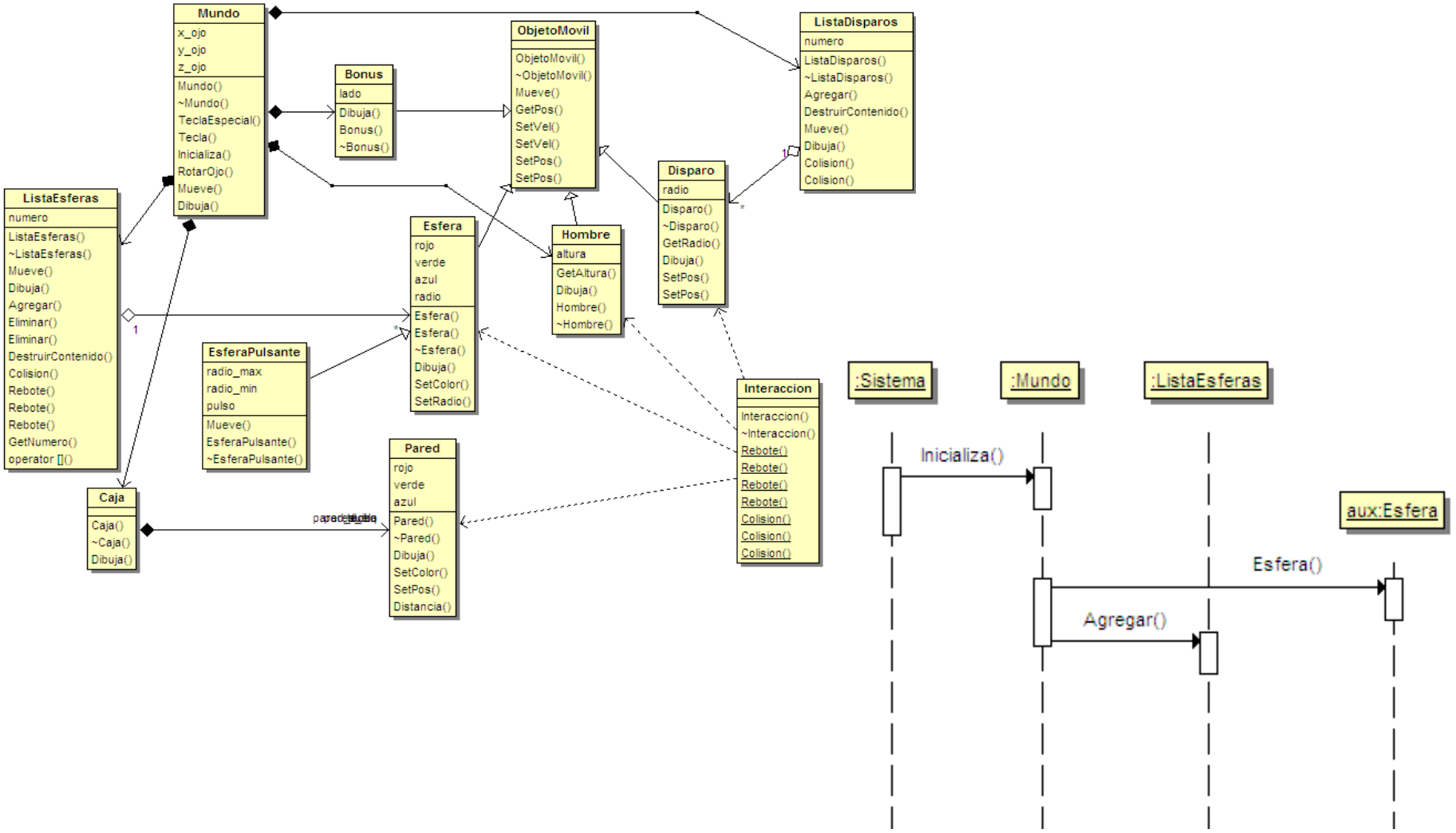
- *Problema:* ¿Quién debería ser el responsable de la creación de una nueva instancia de una clase?
- *Solución:* Asignar a la clase *B* la responsabilidad de crear una instancia de clase *A* si se cumple uno o más de los siguientes casos:
  - B contiene objetos de A
  - B se asocia con objetos de A
  - B registra instancias de objetos de A
  - B utiliza más estrechamente objetos de A
  - B tiene datos de inicialización que se pasarán a un objeto de A
- El patrón creador está relacionado con la asociación y especialmente con la agregación y la composición.
- Relacionado con el patrón Factoría
  - Creación condicional de las instancias

Ejemplo Respuesta Frecuencia

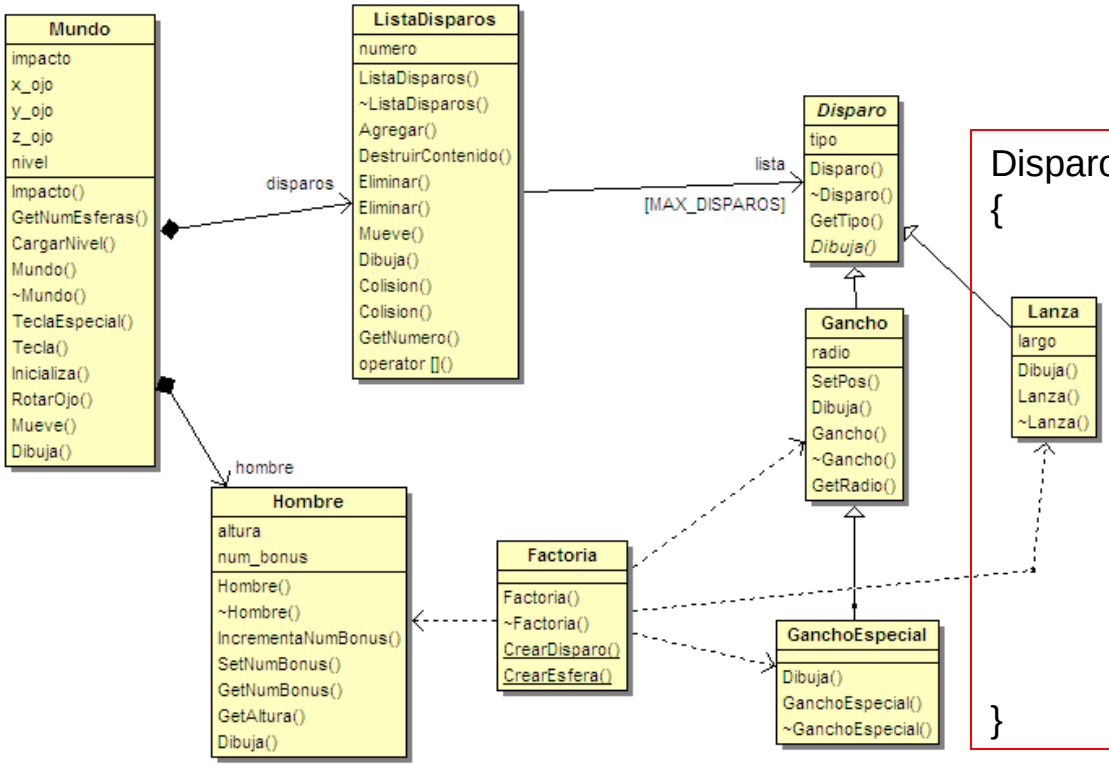



```
#include "../../include/Dominio/CoordinadorFrecELAI.h"
int CoordinadorFrecELAI::ponerCircuito(tipoFiltro elTipo,float resistencia, float condensador)
{
            pFiltro = new FiltroLineal(elTipo,resistencia,condensador);
            return(0);
}
int CoordinadorFrecELAI::ponerParamResFr(float frInicio,float frFinal, float frIntervalo)
{
            if (pFiltro == NULL) return (-1);
            pRespFr = new RespuestaFrecuencia(frInicio,frFinal,frIntervalo,pFiltro);
            return(0);
}
```

Ejemplo de creación: juego del Pang



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;
}
```

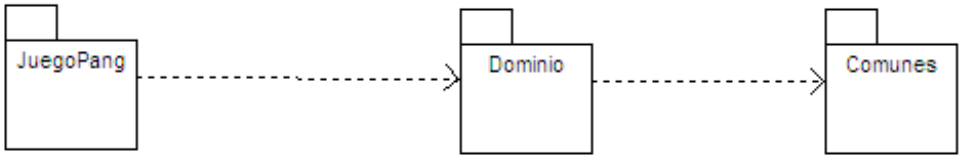
```
void Mundo::Tecla(unsigned char key)
{
  switch(key) {
        case ' ': if(disparos.GetNumero()<MAX_DISPAROS){
                Disparo* d=Factoria::CrearDisparo(hombre);
                disparos.Agregar(d);
                }
                break;
        }
}
```

Alta Cohesión (GRASP)

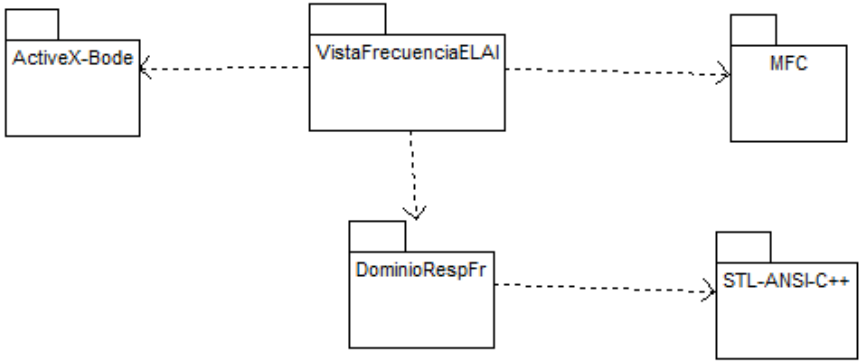
- *Problema*: ¿Cómo mantener la complejidad manejable?
- *Solución*: Asignar una responsabilidad de manera que la cohesión permanezca alta.
- Clases con baja cohesión (hace tareas poco relacionadas o hace mucho trabajo):
  - Difíciles de entender.
  - Difíciles de reutilizar.
  - Difíciles de mantener.
  - Delicadas, constantemente afectadas por los cambios.
- Una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidad altamente relacionada y no realiza mucho trabajo. En el caso de que la tarea sea extensa, colaborará con otros objetos para compartir el esfuerzo.
- Este patrón está relacionado con: Bajo Acoplamiento y Modularidad (alta cohesión y bajo acoplamiento)

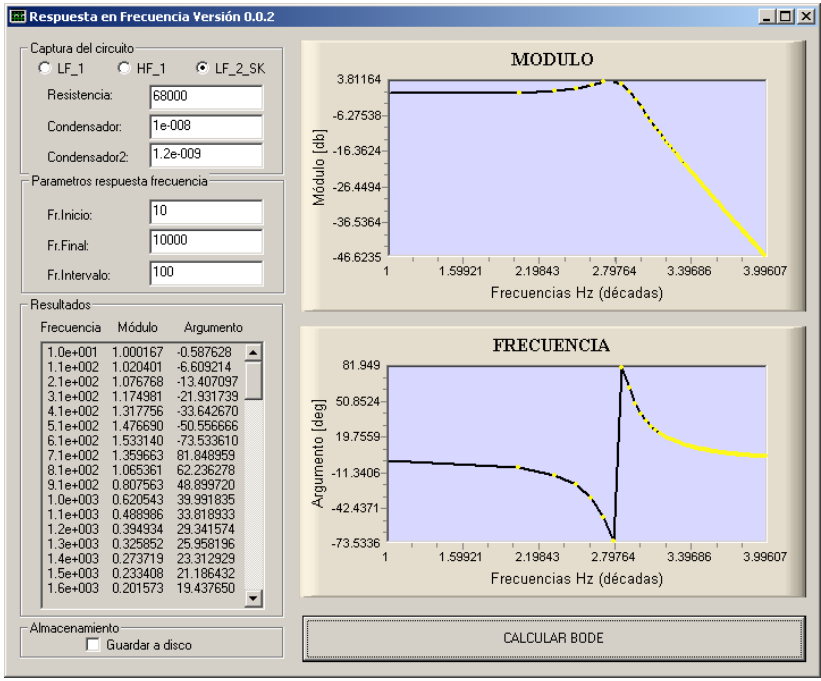
Ejemplos de Alta Cohesión y Bajo Acoplamiento

- Pang:



- Respuesta en frecuencia



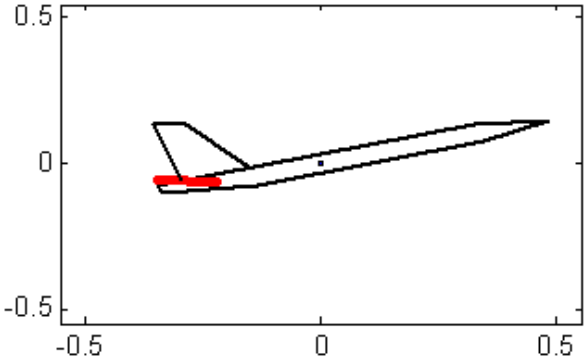


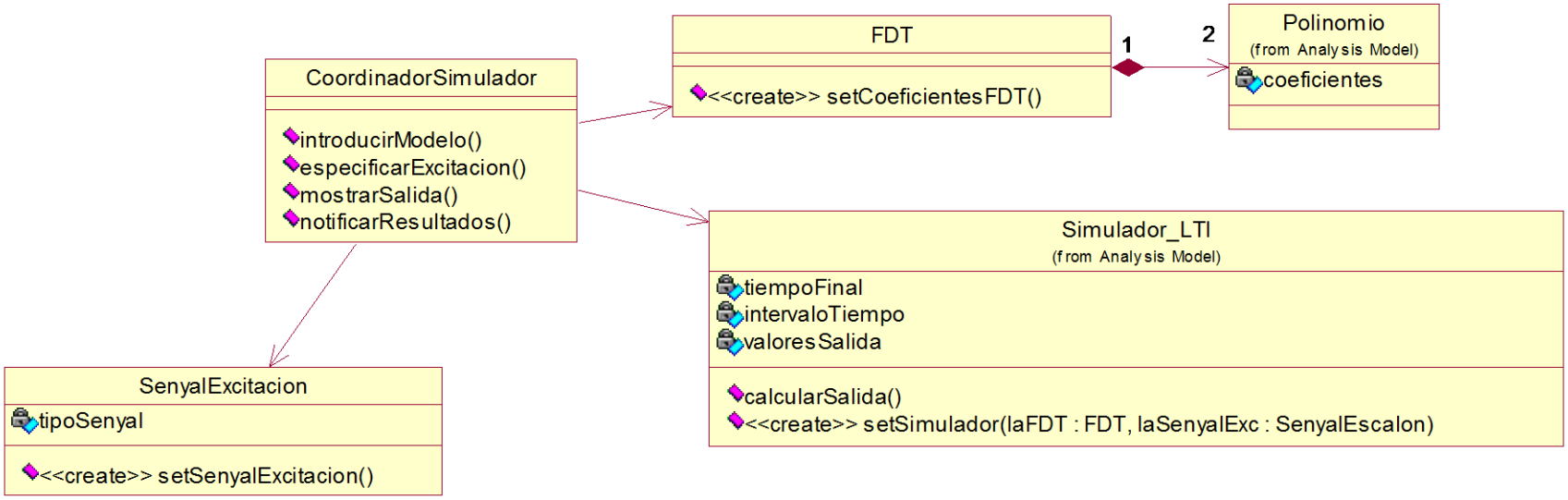
Bajo Acoplamiento (GRASP)

- *Pregunta:* ¿Cómo soportar el bajo impacto del cambio e incrementar la reutilización?
- *Solución:* Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.
- Clases con alto acoplamiento (confía en muchas clases)
  - Son difíciles de mantener de manera aislada.
  - Los cambios en estas clases fuerzan cambios locales.
  - Son difíciles de reutilizar.
- Clases genéricas y con alta reutilización deben tener Bajo Acoplamiento (p.ej: Esfera, Hombre, Disparo)
- Acoplamiento sólo con elementos estables (STL, MFC, Qt,...)

Ejemplo de reutilización

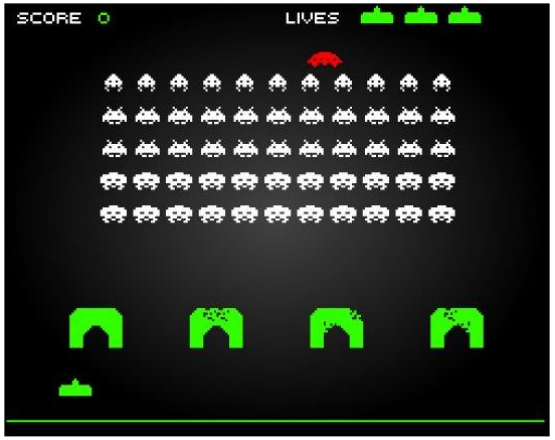
Un simulador de sistemas LTI-SISO requiere para su definición la FDT del sistema. Por tanto, se puede emplear las clases de FDT y Polinomio que se han definido en la Respuesta en Frecuencia para esta otra aplicación. Una definición de clases altamente cohesivas muestra su facilidad de reutilización.

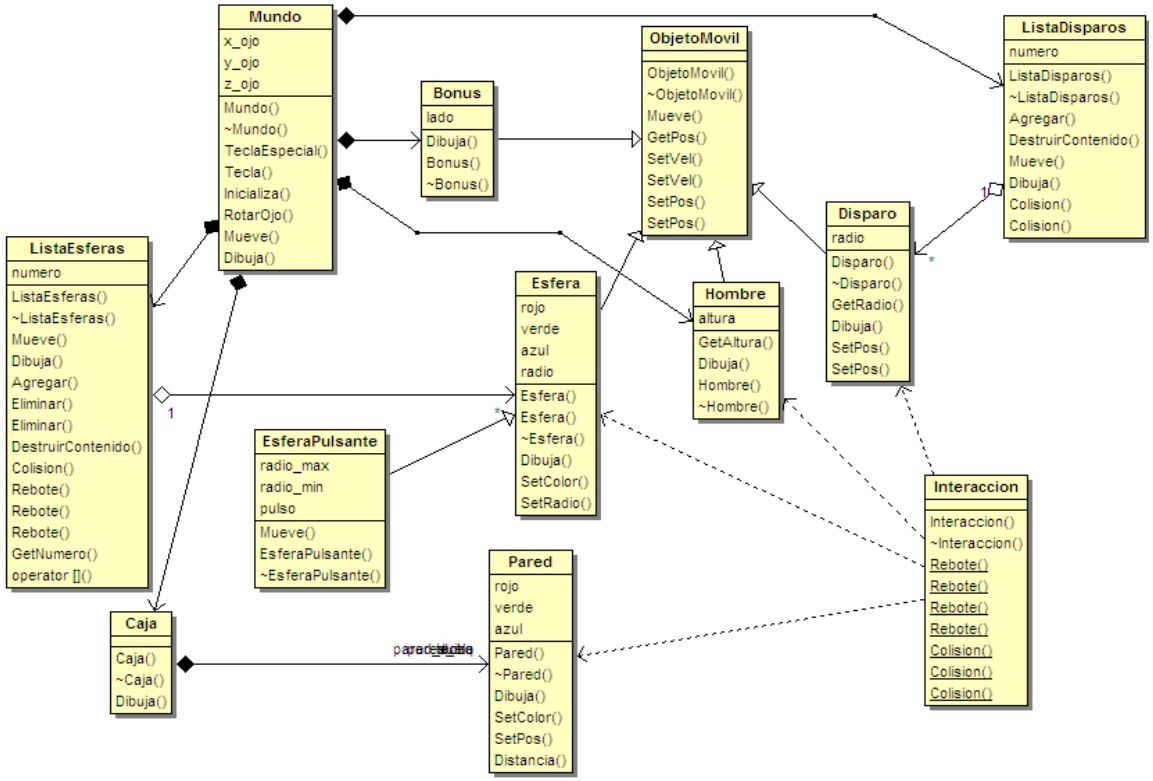




Problema de reutilización

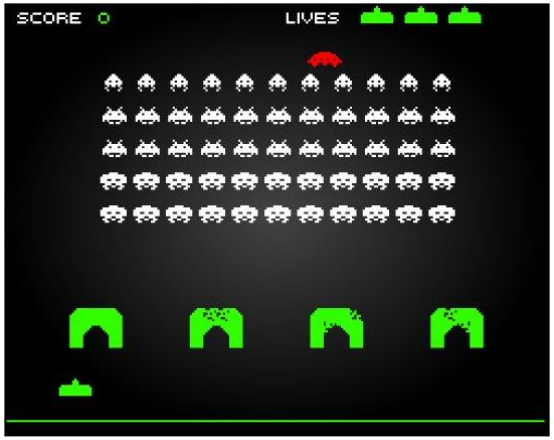
Dado el juego del Pang, reutilizar la clases para el juego de invasión espacial

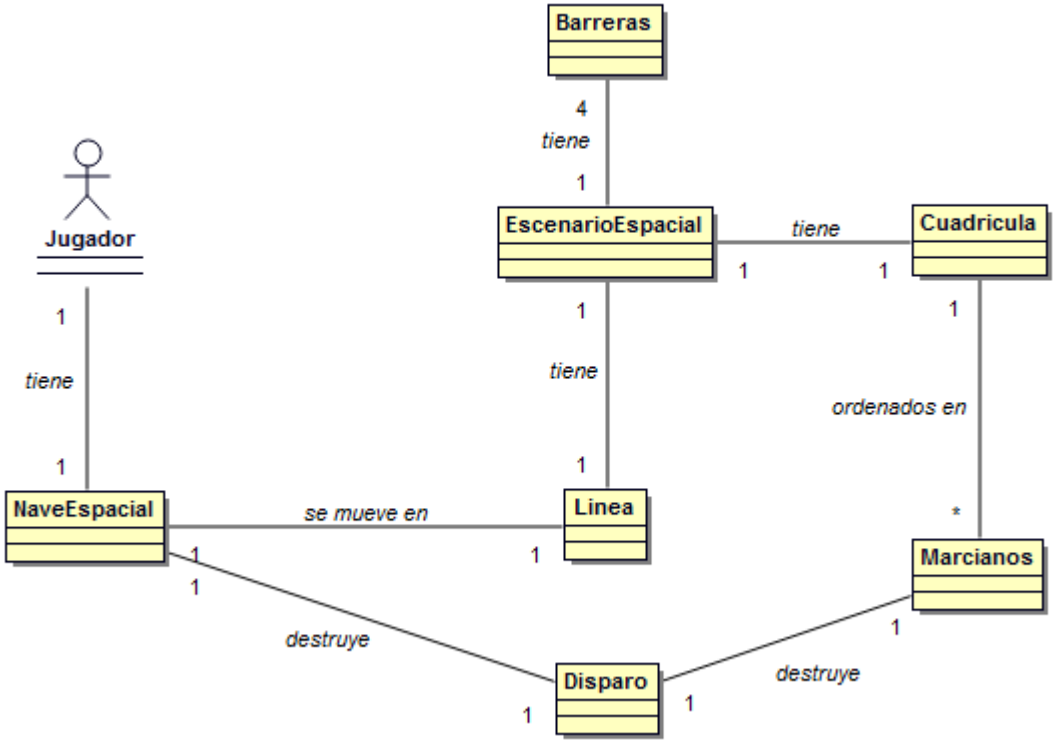




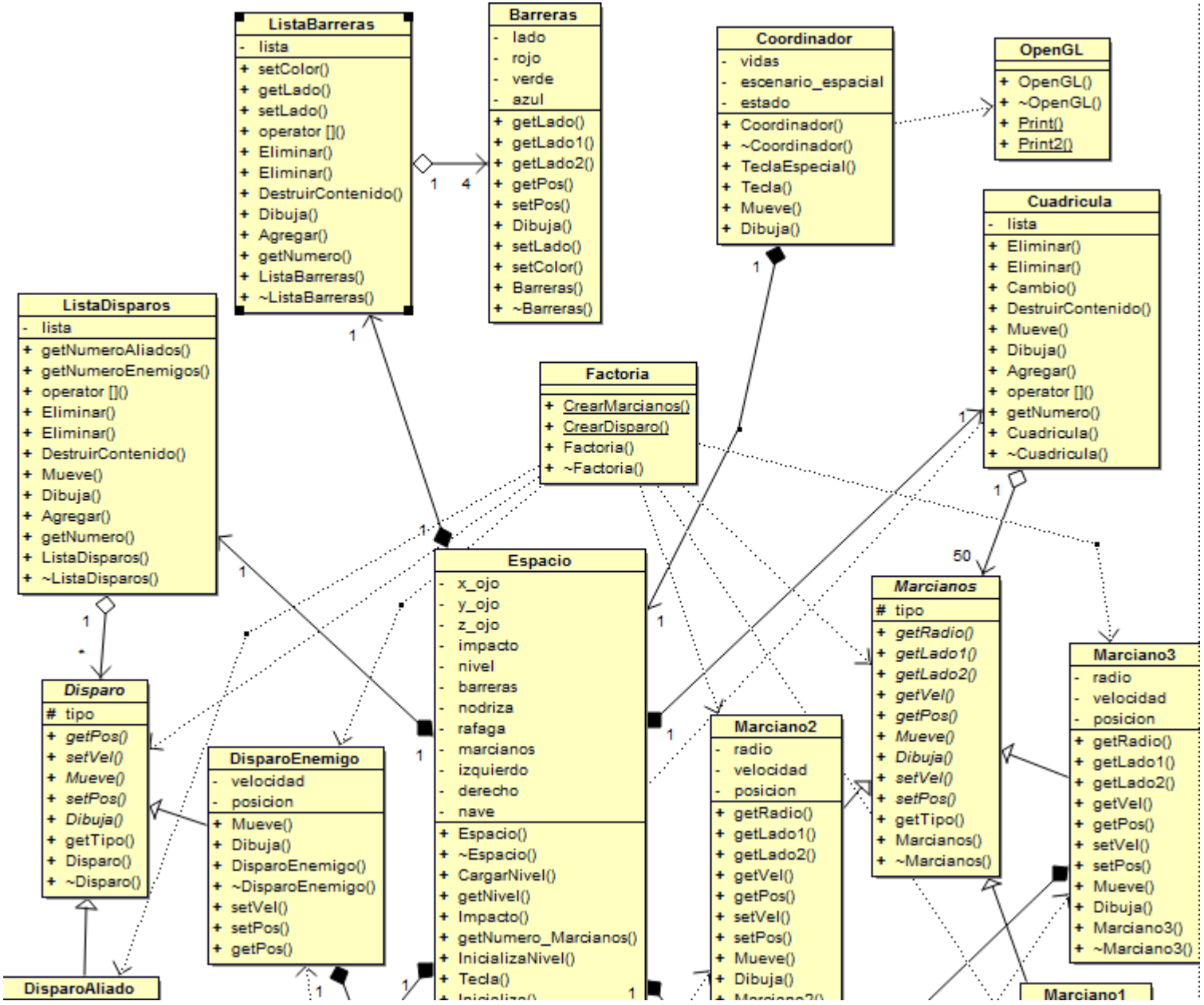
Problema de reutilización

Dado el juego del Pang, reutilizar la clases para el juego de invasión espacial

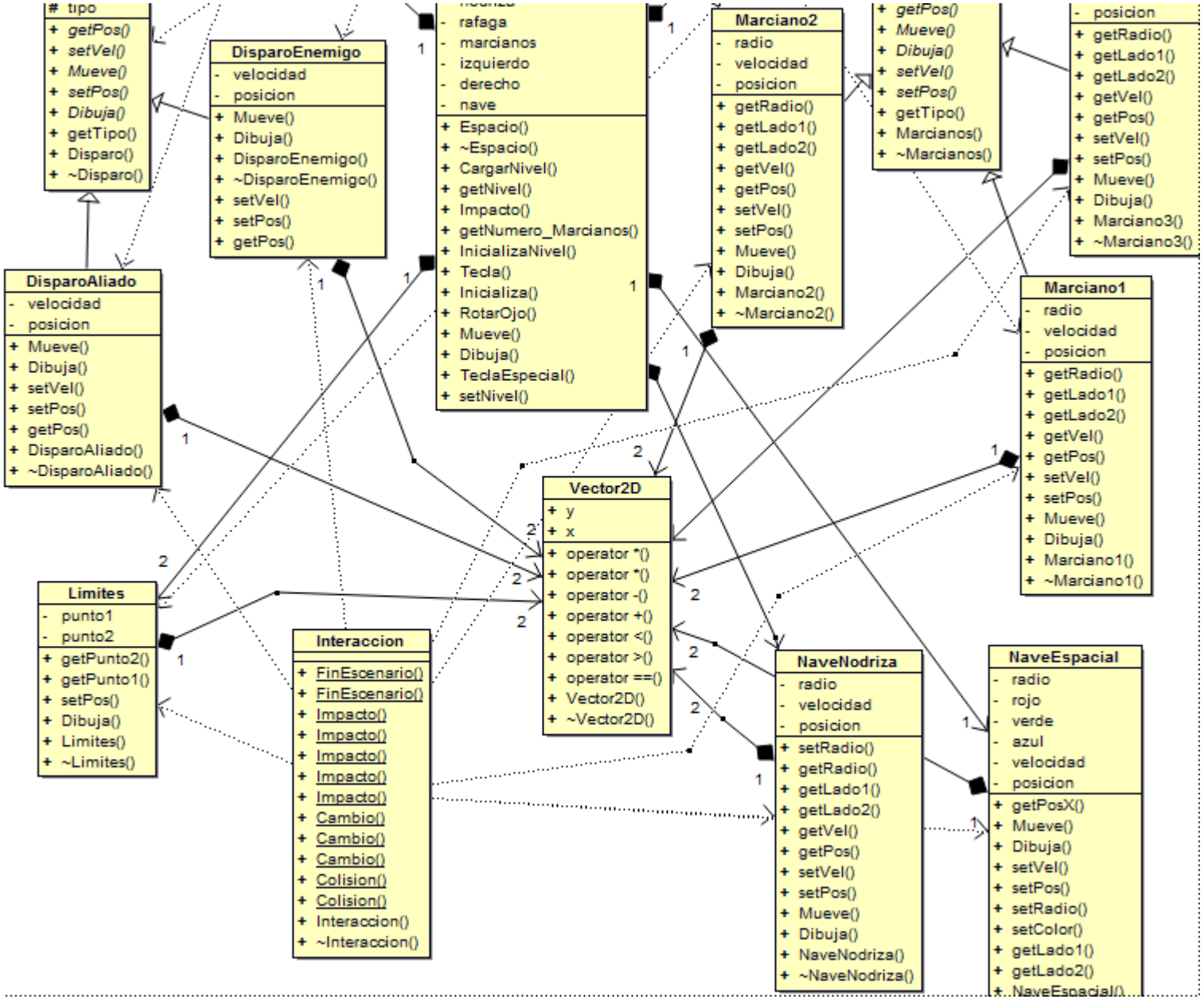




Problema de reutilización



Problema de reutilización



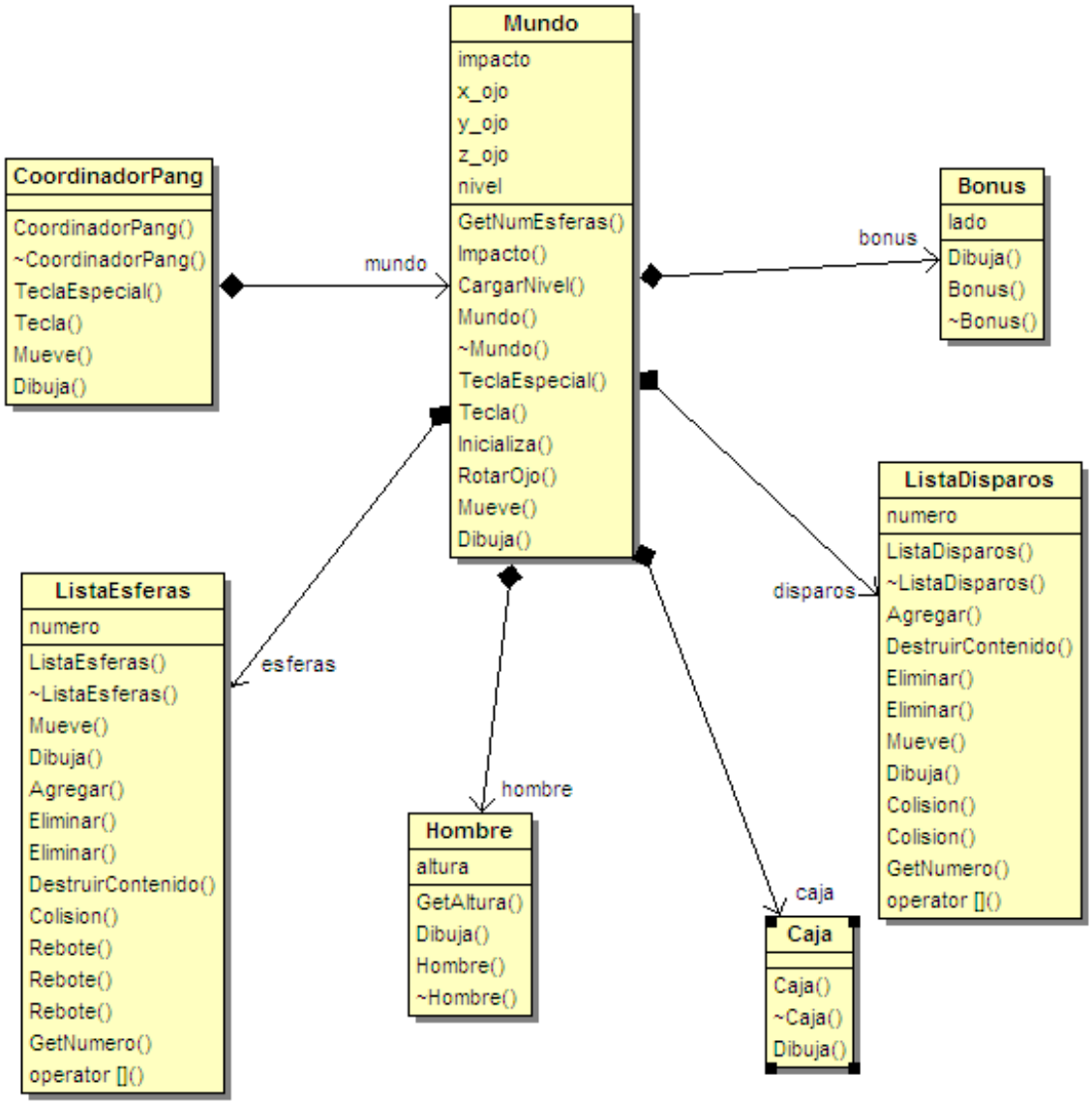
Controlador (GRASP)

- *Problema*: ¿Quién debe ser el responsable de gestionar un evento de entrada al sistema?
- *Solución*: Asignar la responsabilidad a una clase que represente una de las siguientes opciones:
  - Representa el sistema global, dispositivo o subsistema.
  - Representa un escenario de caso de uso.
- Un controlador es un objeto que no pertenece al interfaz o vista, responsable de recibir o manejar los eventos del sistema.
- El controlador es una especie de fachada del paquete que recibe los eventos externos y organiza las tareas.

Controlador (GRASP)

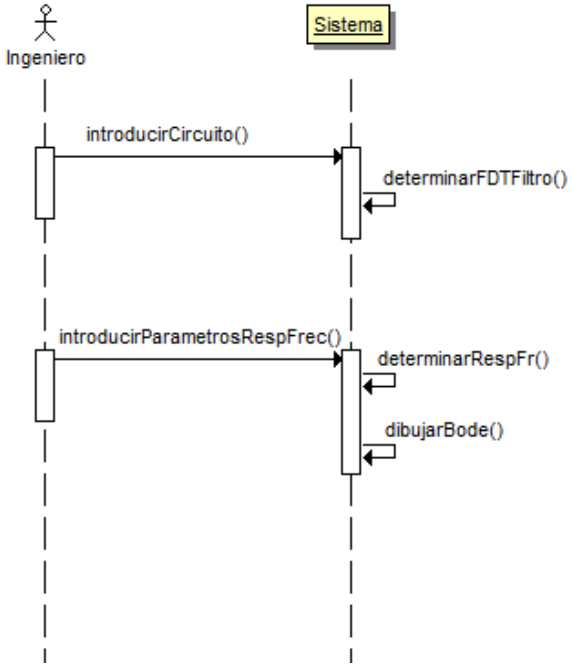
- Un error típico del diseño de los controladores es otorgarles demasiadas responsabilidades.
- El controlador recibe la solicitud del servicio desde una capa superior y coordina su realización, normalmente delegando a otros objetos
- La capa interfaz **no deberían ser responsables de manejar los eventos del sistema.**
- Tipos de objetos:
  - Frontera (interfaz sistema), Control (coordinador), Entidad (dominio)
- El patrón Controlador crea un objeto artificial que no procede del análisis del dominio. Se dice que es una Fabricación Pura.

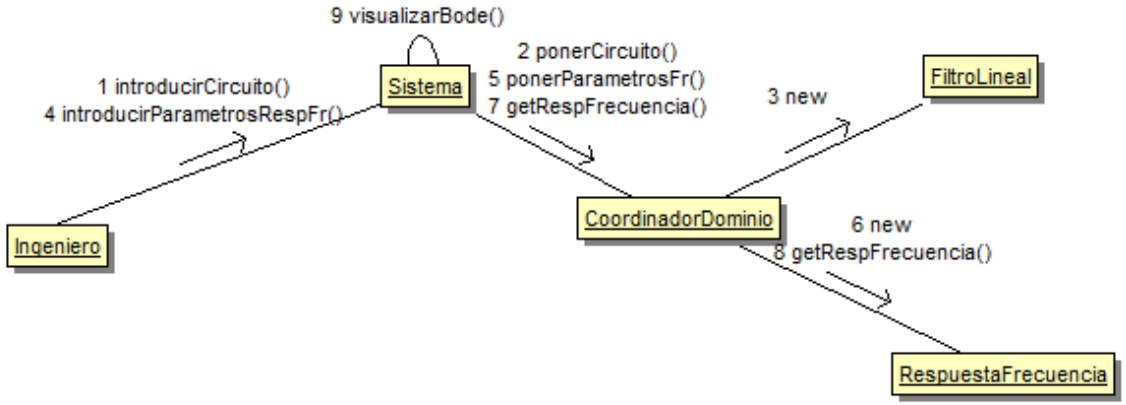
Ejemplo Pang



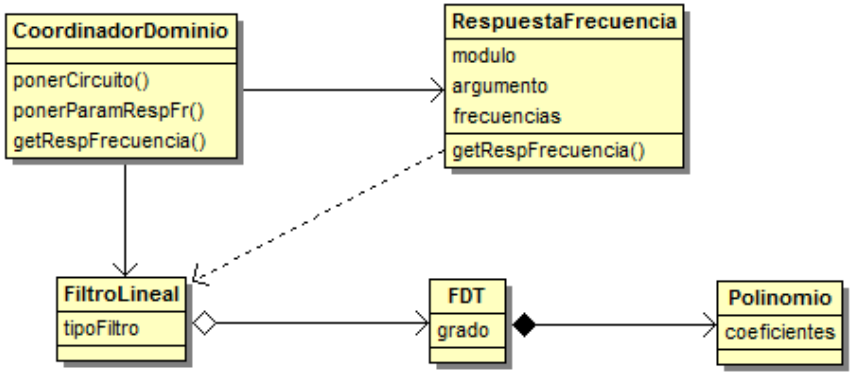
Ejemplo Respuesta Frecuencia

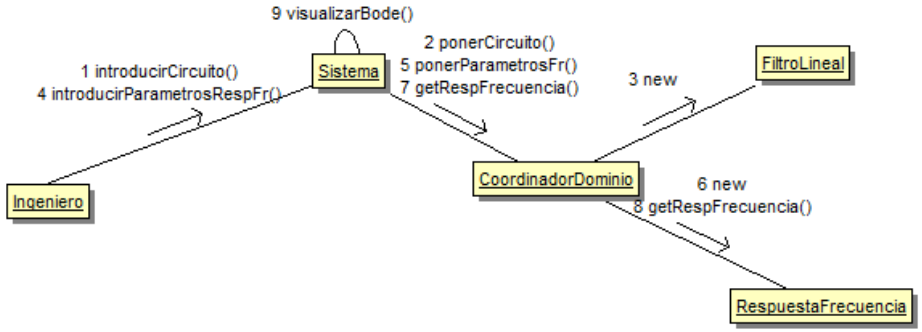
Emplear un Coordinador para la aplicación de Respuesta en Frecuencia que organice las tareas del escenario de caso de uso y que delegue las tareas.





Ejemplo Respuesta Frecuencia






```
#include "../../include/Dominio/CoordinadorFrecELAI.h"
int CoordinadorFrecELAI::ponerCircuito(tipoFiltro elTipo,float resistencia, float condensador)
{
            pFiltro = new FiltroLineal(elTipo,resistencia,condensador);
            return(0);
}
int CoordinadorFrecELAI::ponerParamResFr(float frInicio,float frFinal, float frIntervalo)
{
            if (pFiltro == NULL) return (-1);
            pRespFr = new RespuestaFrecuencia(frInicio,frFinal,frIntervalo,pFiltro);
            return(0);
}
int CoordinadorFrecELAI::getModuloRespFr(std::vector<double> &elVectorModulo)
{
            if (pRespFr == NULL) return (-1);
            pRespFr->getModuloRespFr(elVectorModulo);
            if(pFiltro) delete pFiltro;
            if(pRespFr) delete pRespFr;
            return (0);
}
```

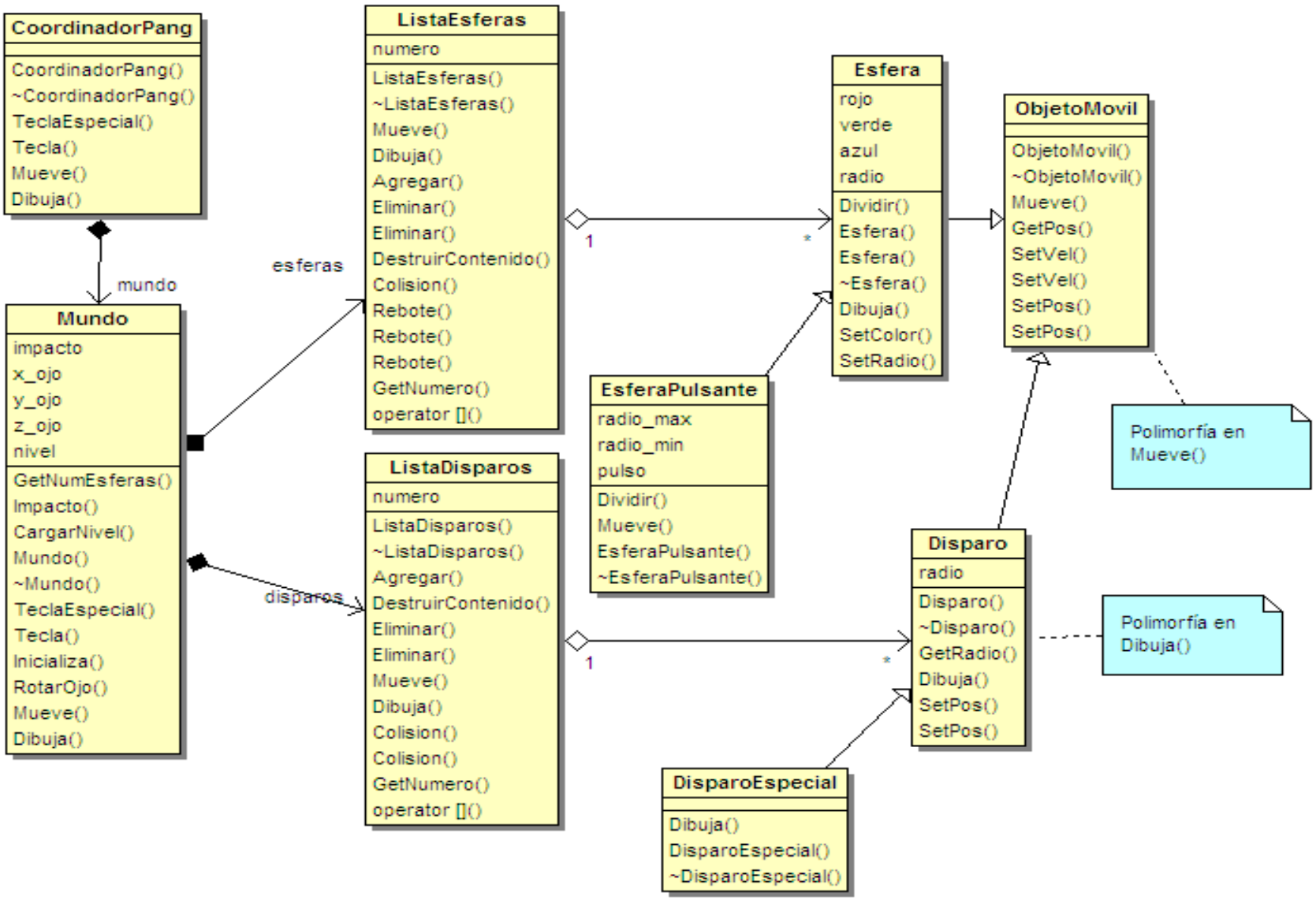
Polimorfismo (GRASP)

- *Problema*: ¿Cómo manejar las alternativas basadas en tipo?, ¿Cómo crear componentes software conectables (*pluggable*)? o ¿Cómo se puede sustituir un componente servidor por otro, sin afectar al cliente?
- *Solución*: Cuando las alternativas o comportamientos relacionados varían según los tipos de los datos, se debe asignar la responsabilidad utilizando operaciones polimórficas, de forma que varía el comportamiento según el tipo. No hay que realizar comprobaciones acerca del tipo del objeto. No se requiere emplear la lógica condicional para llevar a cabo alternativas diferentes basadas en el tipo.

Polimorfismo (GRASP)

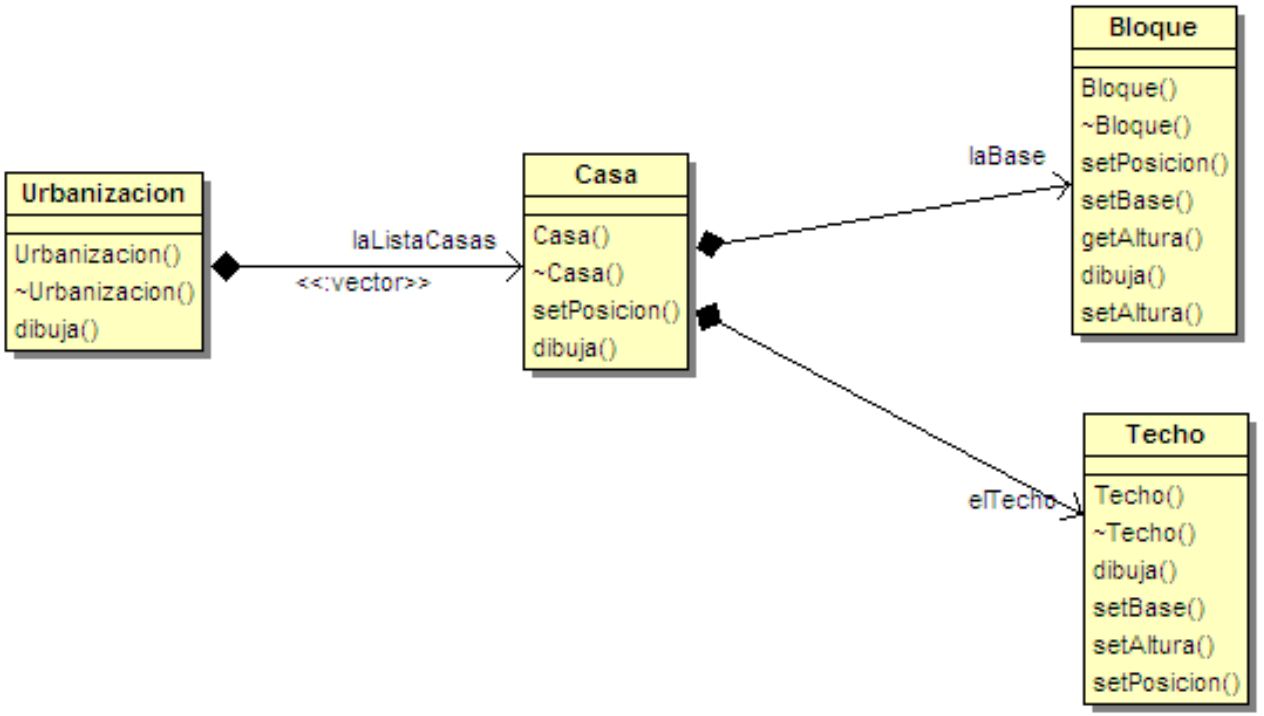
- Un diseño que emplea las bifurcaciones *if-else* o *switch-case*, en cada nueva variación requiere la modificación de esta lógica. Este enfoque dificulta que el programa se extienda con facilidad.
- El polimorfismo es un principio fundamental para designar cómo se organiza el sistema para gestionar variaciones similares
- DOO: Interface + Polimorfismo

Ejemplo: Pang

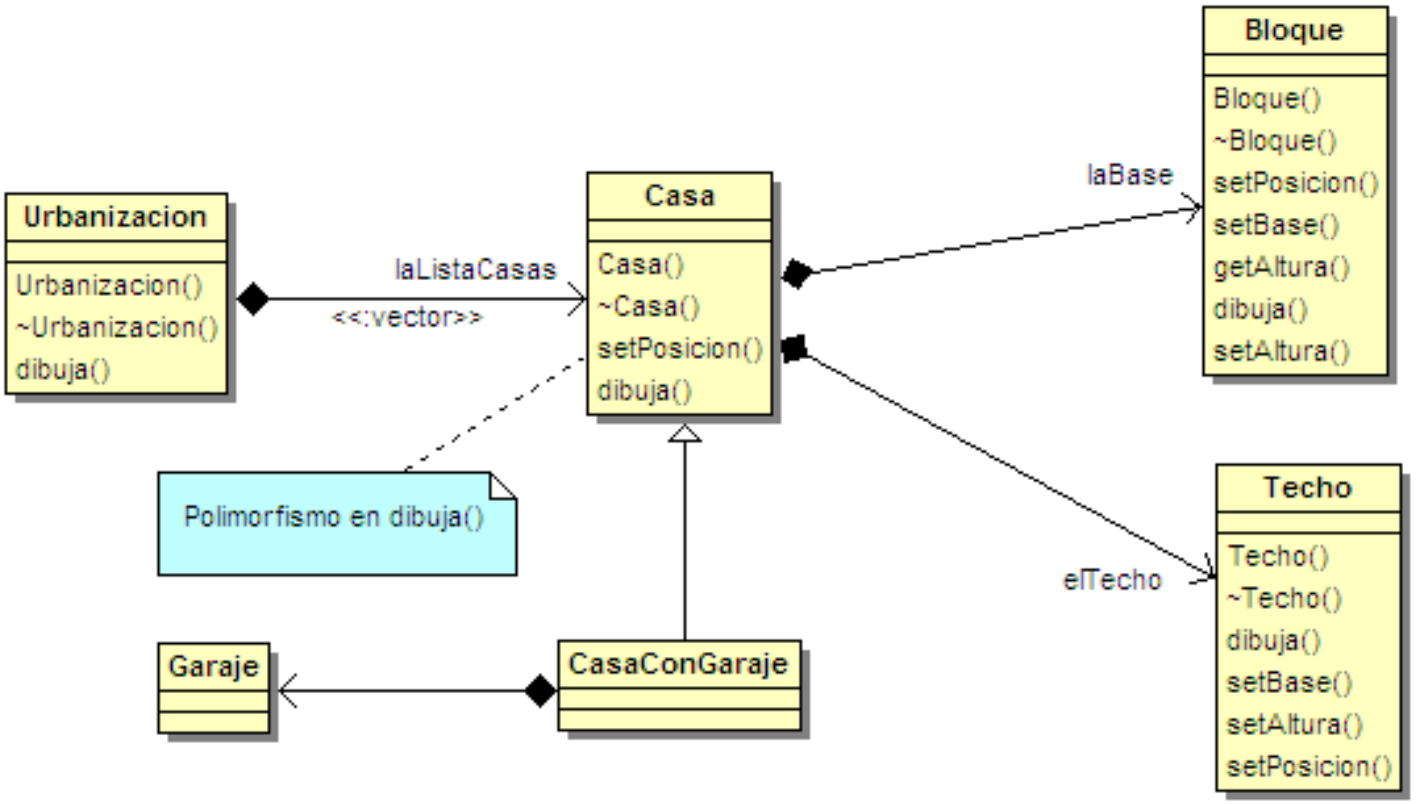


Ejemplo: Urbanización

En el ejemplo de la urbanización, ¿cómo asignaría la responsabilidad de una casa con garaje?



Ejemplo: Urbanización



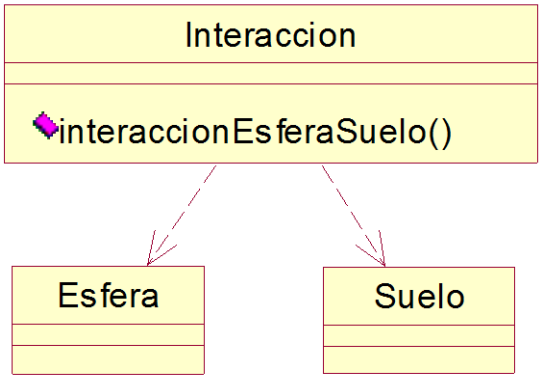
Indirección (GRASP)

- *Problema*: ¿Donde asignar una responsabilidad, para evitar el acoplamiento directo entre dos o más lógicas de la aplicación?, ¿Cómo desacoplar los objetos de manera que se soporte el Bajo Acoplamiento y el potencial de reutilización permanezca alto?.
- *Solución*: Asignar la responsabilidad a un objeto intermedio entre dos o más elementos o paquetes de manera que no se acoplen directamente.
- La mayoría de los intermediarios de Indirección son Fabricaciones Puras.

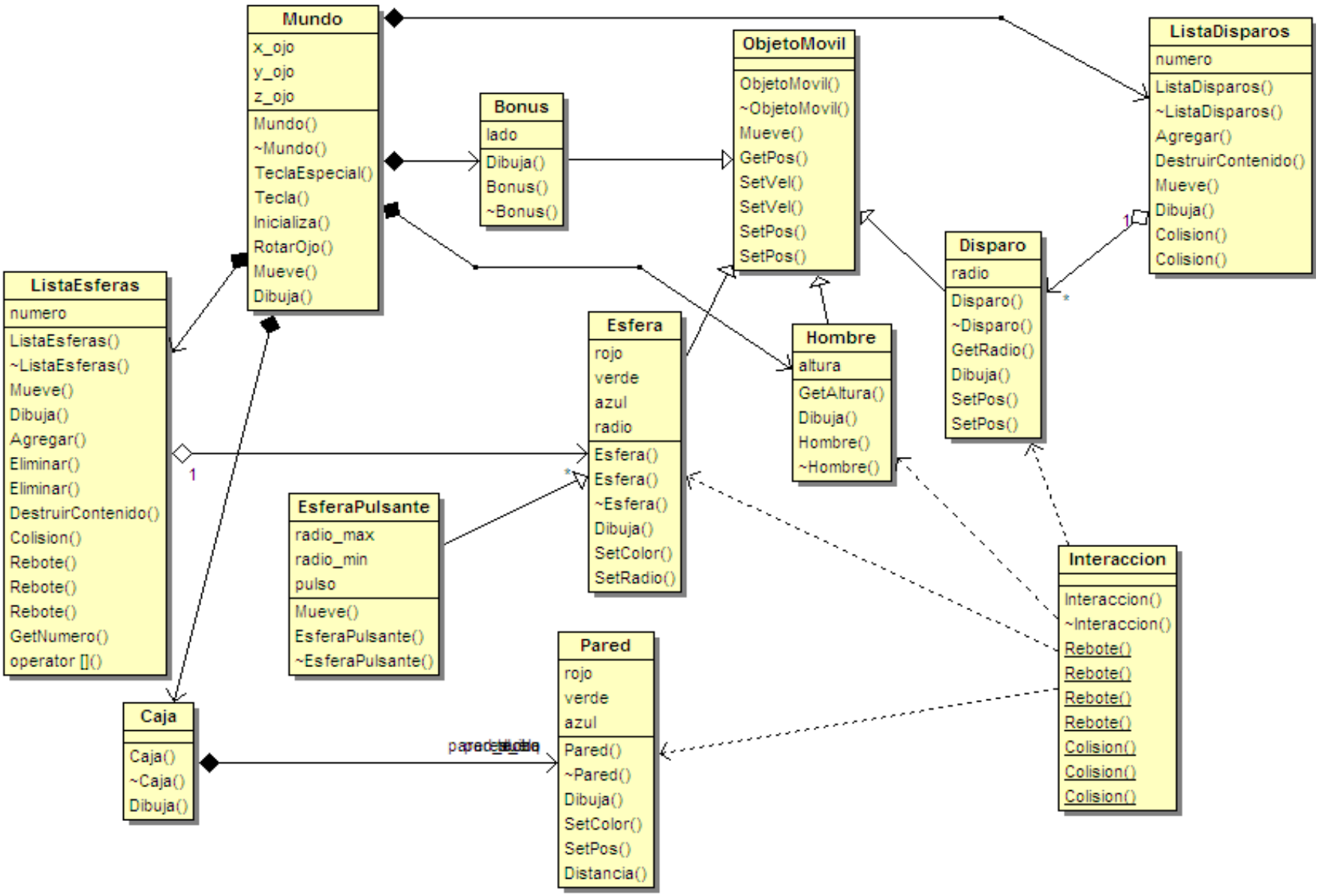
Ejemplo de Indirección

En los programas de simulación donde los objetos cambian de dinámica al chocar con otros objetos, ¿cómo se asignarían las responsabilidades?. Supóngase que se desea simular cómo una esfera en caída libre es arrojada desde una cierta altura respecto al suelo.

Suelo y Esfera son clases conceptuales y por tanto candidatas a ser clases de diseño. Para evitar el acoplamiento entre ambas clases se añade un grado de *indirección*. Se creará una clase interacción que resuelva la responsabilidad de la interacción entre las instancias de las dos clases.



Juego del Pang



Fabricación Pura (GRASP)

- *Problema*: ¿Qué objetos deberían de tener las responsabilidades cuando no se quiere romper los objetivos de Alta Cohesión y Bajo Acoplamiento, pero las soluciones que ofrece el Experto no son adecuadas?
- *Solución*: Asignar responsabilidades altamente cohesivas a una clase artificial o de conveniencia que no represente un concepto del dominio del problema. Algo inventado para soportar Alta Cohesión, Bajo Acoplamiento y Reutilización.

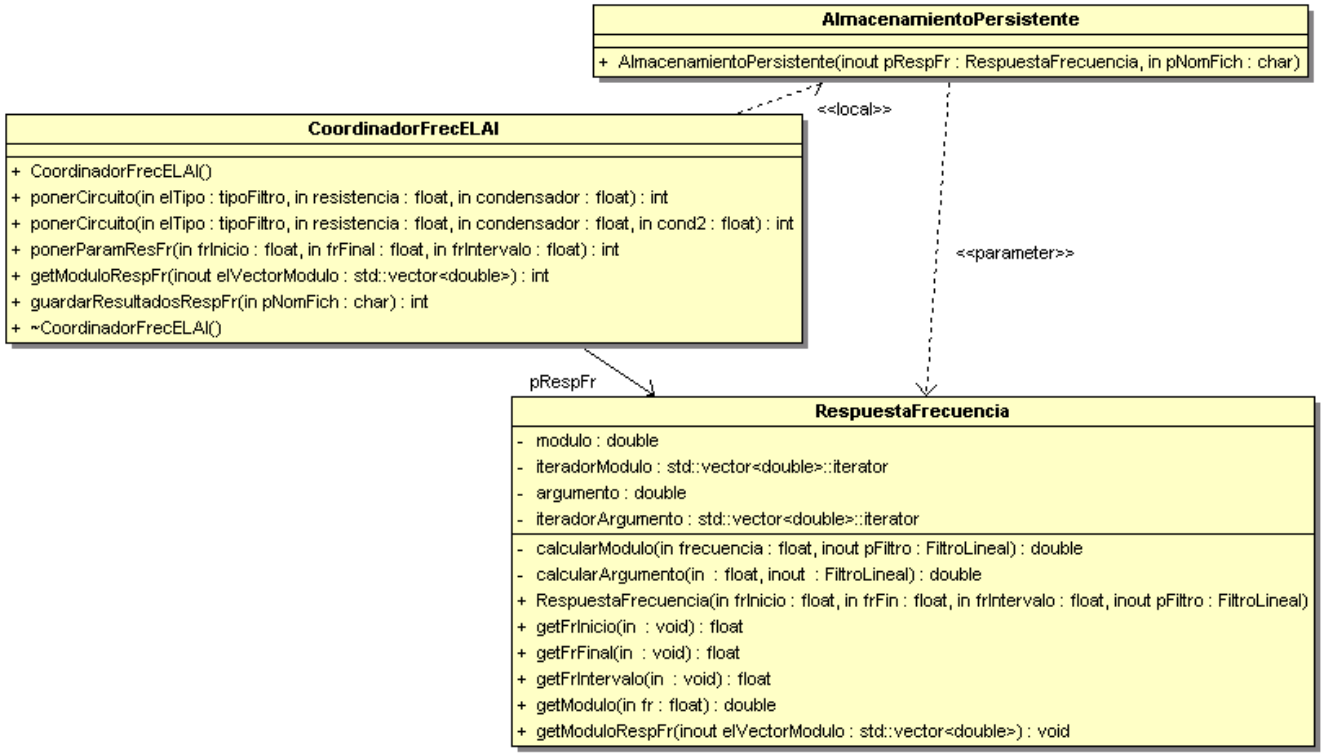
Fabricación Pura (GRASP)

- El diseño de objetos se puede dividir, en general, en dos grandes grupos
  - Los escogidos de acuerdo a una descomposición de la representación (*Polinomio, FiltroLineal, FDT,...*).
  - Los escogidos según una descomposición del comportamiento (*CoordinadorRespFr*).
- Fabricación Pura asume responsabilidades de las clases del dominio a las que se les asignaría esas responsabilidades en base al patrón Experto; pero que no se las da, debido a que disminuiría en cohesión y aumentaría la dependencia.
- La Fabricación Pura emplea el patrón de Indirección

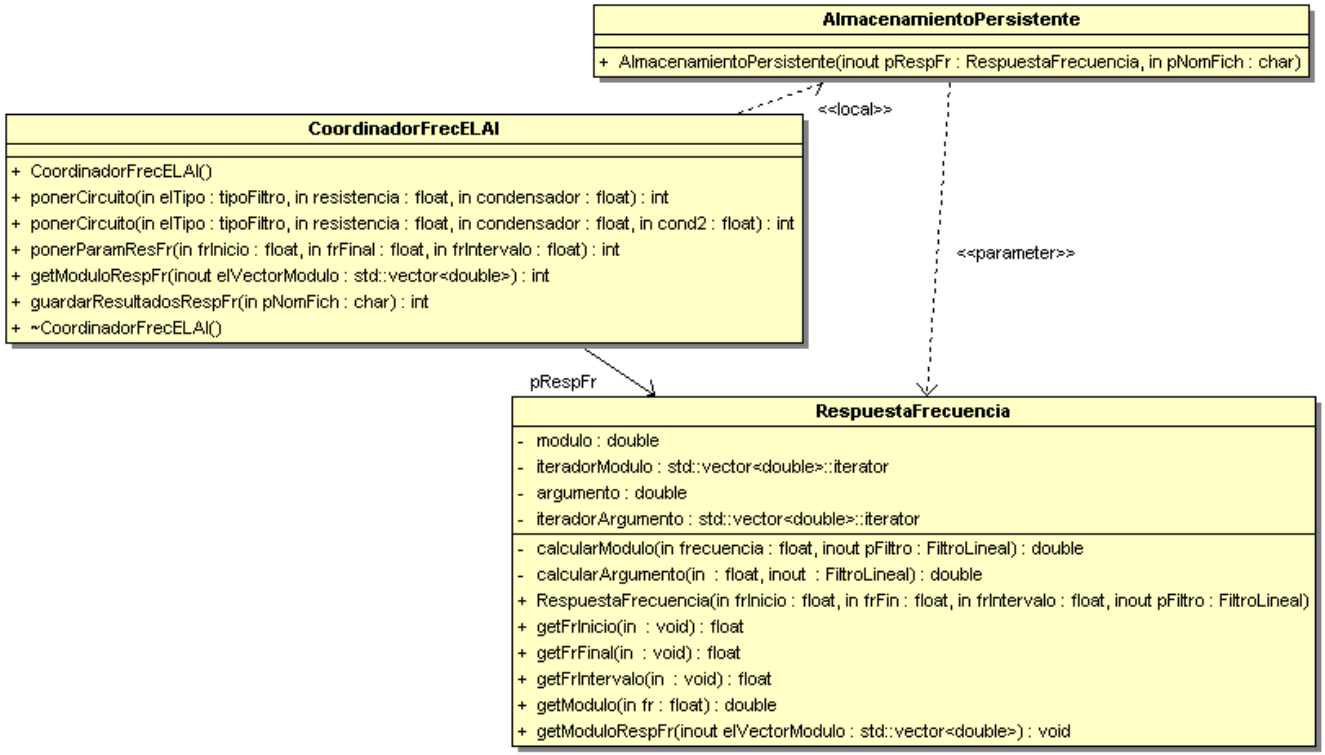
Ejemplo de Respuesta en Frecuencia

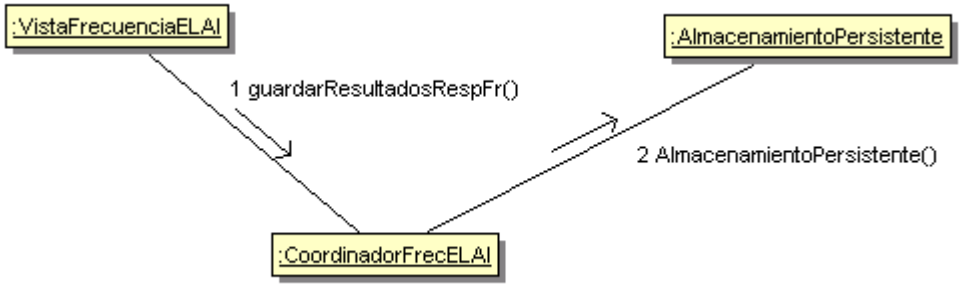
Guardar la información de la respuesta en frecuencia en un fichero.

Para mantener alta la cohesión y no romper la lógica del dominio con el de la base de datos se emplea una Fabricación Pura. Empleando un grado de Indirección se introduce la clase *AlmacenamientoPersistente*. Ésta se encargará de guardar la información en disco. También habrá que añadir este nuevo servicio al Coordinador.



Ejemplo de Respuesta en Frecuencia





Ejemplo de Respuesta en Frecuencia

```
// De: "Apuntes de Sistemas Informáticos Industriales" Carlos Platero.
// Ver permisos en licencia de GPL
#include "../../include/Dominio/AlmacenamientoPersistente.h"
AlmacenamientoPersistente::AlmacenamientoPersistente(RespuestaFrecuencia *pRespFr,
      const char * pNomFich)
{
     ofstream os(pNomFich);
     os <<"Modulo de la respuesta en frecuencia"<<endl;
     float fr;
     for (fr = pRespFr->getFrInicio();fr <= pRespFr->getFrFinal();
               fr+=pRespFr->getFrIntervalo())
                os << fr << " :" << pRespFr->getModulo(fr)<<endl;
}
int CoordinadorFrecELAI::guardarResultadosRespFr(const char *pNomFich)
{
     if (pRespFr == NULL) return (-1);
     AlmacenamientoPersistente elAlmacen(this->pRespFr,pNomFich);
     return (0);
}
```

```
f:\cpd\InfoInd\Practicas\pr11RespFrv2\ResFrConsolav2\Debug\FrecuenciaELAI.exe
Valor de la resistencia: 68e3

Valor del condensador : 1e-8

Valor del condensador2: 1.2e-9

Cual es la frecuencia inicial [Hz]: 10

Cual es la frecuencia final [Hz]: 1e4

Cual es el intervalo empleado para el cβlculo [Hz]: 1e3
1.00017
0.620543
0.123209
0.0523615
0.0290099
0.0184407
0.0127605
0.00935561
0.00715359
0.00564738

Desea guardar los resultados (s/n):s

Nombre del fichero: filtro2.txt_
```

Problema de examen

Para el código adjuntado se pide:

1. Ingeniería inversa: diagrama de clases.
2. Ingeniería inversa: diagrama de secuencia de la función ***main()***.
3. Resultado de su ejecución en la consola.
4. Diseñar e implementar el servicio rotar( ), tal que

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

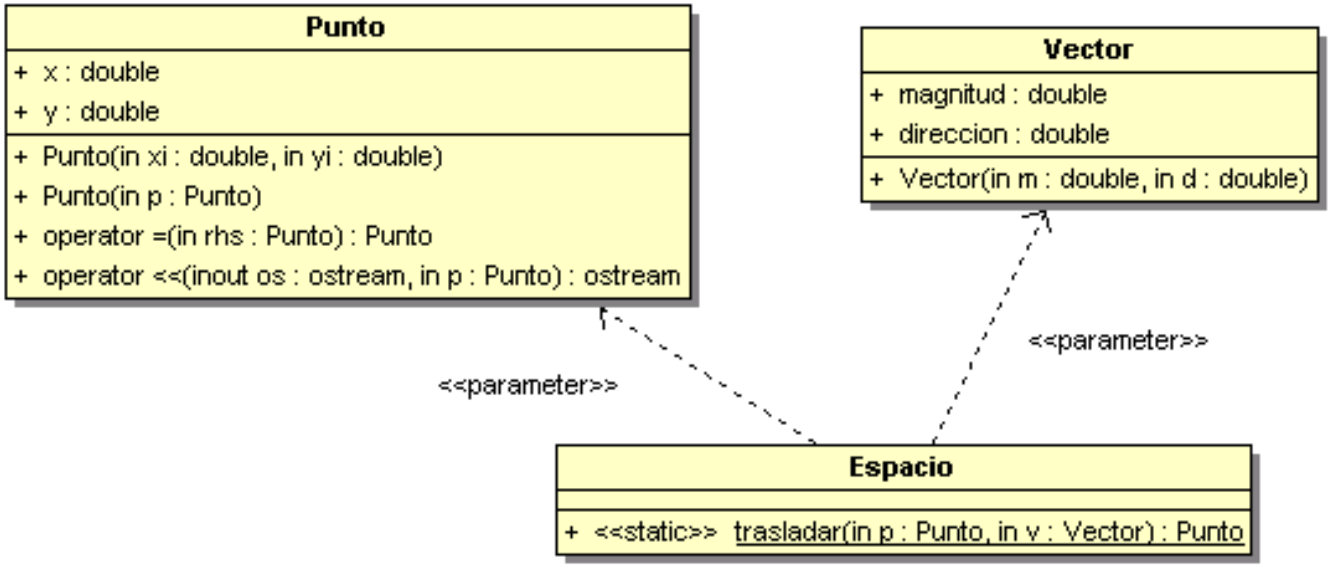
Empléese sobre el punto p3.

Problema de examen

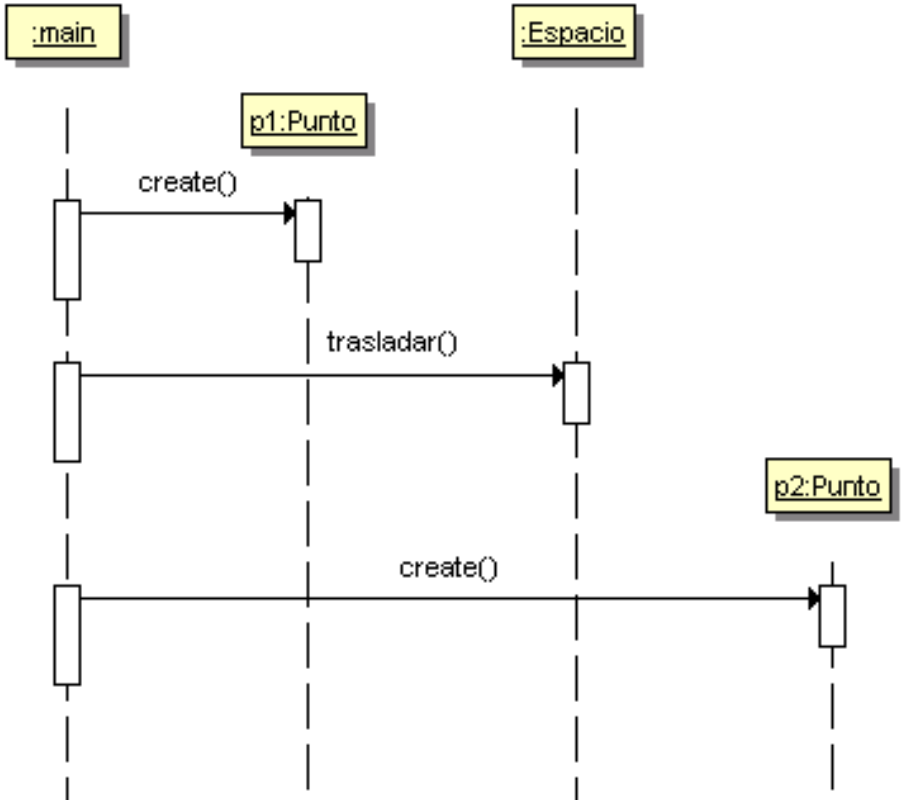
```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

class Punto {
public:
  double x, y;
  Punto(double xi, double yi) : x(xi), y(yi) {}
  Punto(const Punto& p) :  x(p.x), y(p.y) {}
  Punto& operator=(const Punto& rhs) {
    x = rhs.x;
    y = rhs.y;
    return *this;
  }
  friend ostream&
  operator<<(ostream& os, const Punto& p) {
    return os << "x=" << p.x << " y=" << p.y;
  }
};

class Vector {
public:
  double magnitud, direccion;
  Vector(double m, double d) : magnitud(m),
direccion(d) {}
};
```

```
class Espacio {
public:
  static Punto trasladar(Punto p, Vector v) {
    p.x += (v.magnitud * cos(v.direccion));
    p.y += (v.magnitud * sin(v.direccion));
    return p;
  }
};

int main() {
  Punto p1(1, 2);
  Punto p2 = Espacio::
          trasladar(p1, Vector(3, 3.1416/3));
  cout << "p1: " << p1 << " p2: " << p2
       << endl;

  return 0;
}
```

Problema de examen



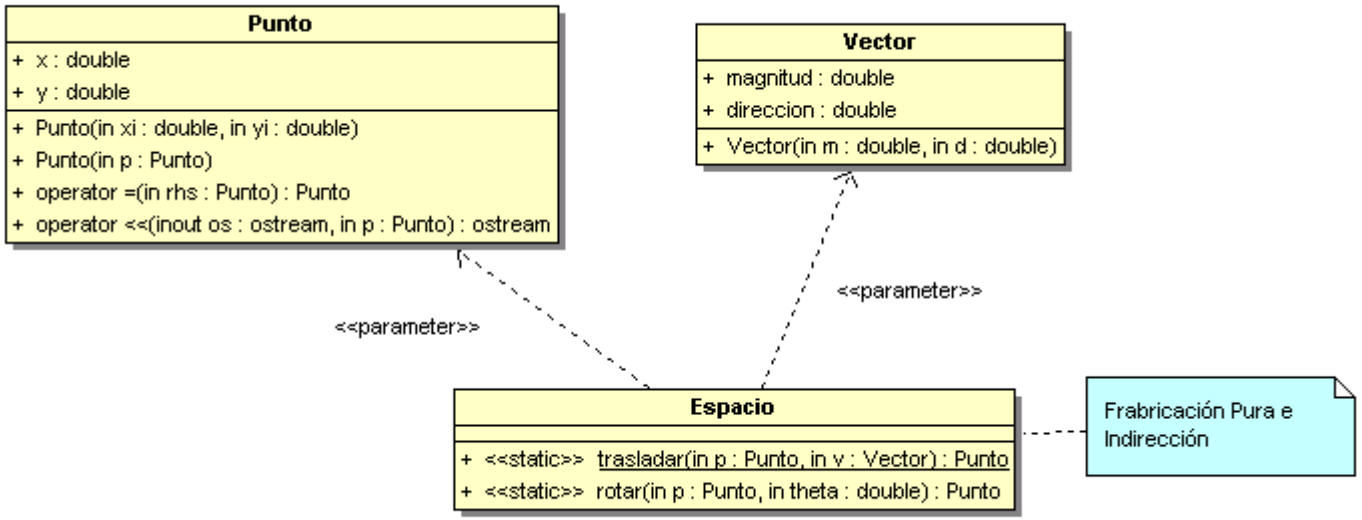
Problema de examen



Problema de examen

3) p1: x=1 y=2 p2: x=2.5 y=4.6

4) Se ha aplicado Experto de Información en la clase Punto y Vector. Para evitar el acoplamiento entre ambas clases se ha aplicado el patrón Indirección y por tanto una Fabricación Pura con la clase Espacio. El servicio rotar() será responsabilidad de la clase Espacio.



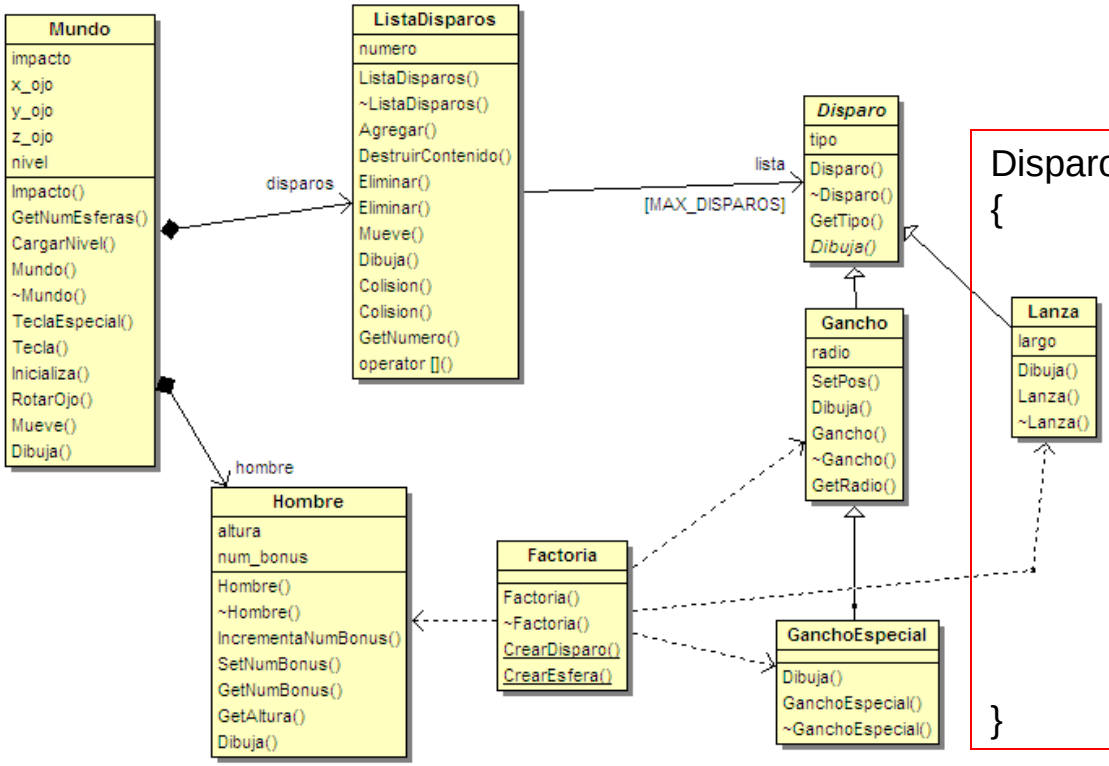
Problema de examen

```
class Espacio {
public:
  static Punto trasladar(Punto p, Vector v) {
    p.x += (v.magnitud * cos(v.direccion));
    p.y += (v.magnitud * sin(v.direccion));
    return p;
  }
  static Punto rotar(Punto p, double theta) {
    Punto res(0,0);
    res.x = (p.x * cos(theta)) - (p.y *sin(theta));
    res.y = (p.x * sin(theta)) + (p.y *cos(theta));
    return res;
  }
};

int main() {
  Punto p1(1, 2);
  Punto p2 = Espacio::trasladar(p1, Vector(3, 3.1416/3));
  Punto p3 = Espacio::rotar(p2,3.1416/6);

  cout << "p1: " << p1 << " p2: " << p2 << " p3: " << p3 <<endl;

  return 0;
}
```

Variaciones Protegidas(1/2)

- *Problema*: ¿Cómo diseñar objetos, subsistemas y sistemas de manera que las variaciones e inestabilidades en estos elementos no tengan un impacto negativo en otros elementos?
- *Solución:* Identifique los puntos de variaciones previstas e inestabilidad; asigne responsabilidades para crear una interfaz estable alrededor de ellos. Añadiendo ***Indirección, Polimorfismo y una interfaz*** se consigue un sistema de Variaciones Protegidas, VP. Las distintas implementaciones del componente y/o paquete ocultan las variaciones internas a los sistemas clientes de éste. Dentro del componente, los objetos internos colaboran en sus tareas con una interfaz estable.
- Hay que distinguir dos tipos de variaciones:
  - Puntos de variación: variaciones en el sistema actual.
  - Puntos de evolución: puntos especulativos de variación que podrían aparecer en el futuro, pero que no están presentes en los requisitos actuales.

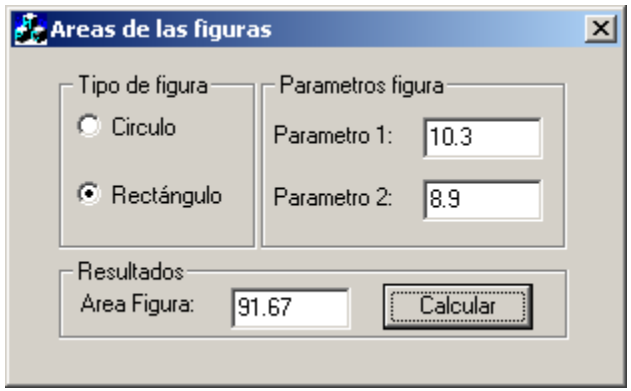
Ejemplo: Juego del Pang

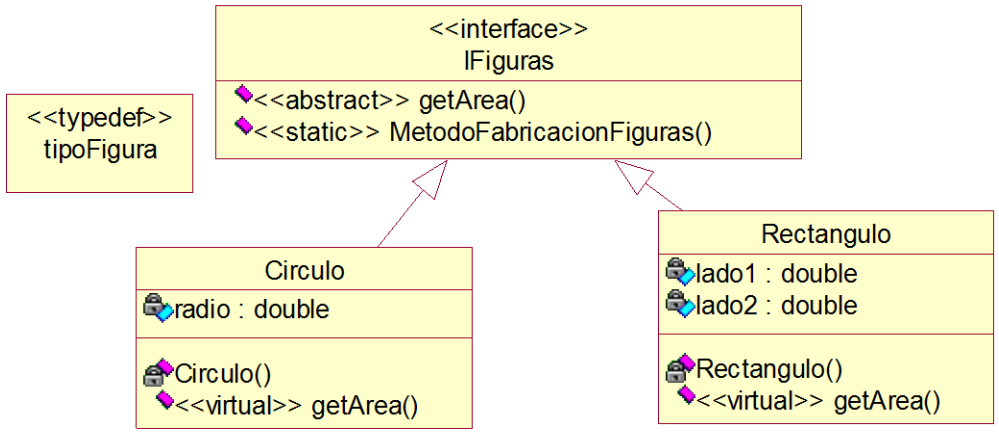



```
Disparo* Factoria::CrearDisparo(Hombre h)
{
        Disparo* d=0;

        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) {
        case ' ': if(disparos.GetNumero()<MAX_DISPAROS){
                Disparo* d=Factoria::CrearDisparo(hombre);
                disparos.Agregar(d);
                }
                break;
        }
}
```

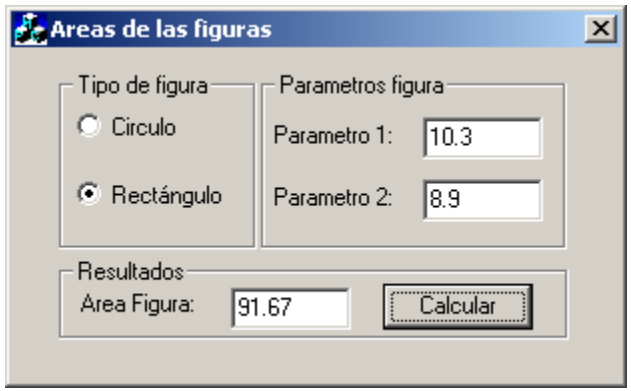
Ejemplos VP

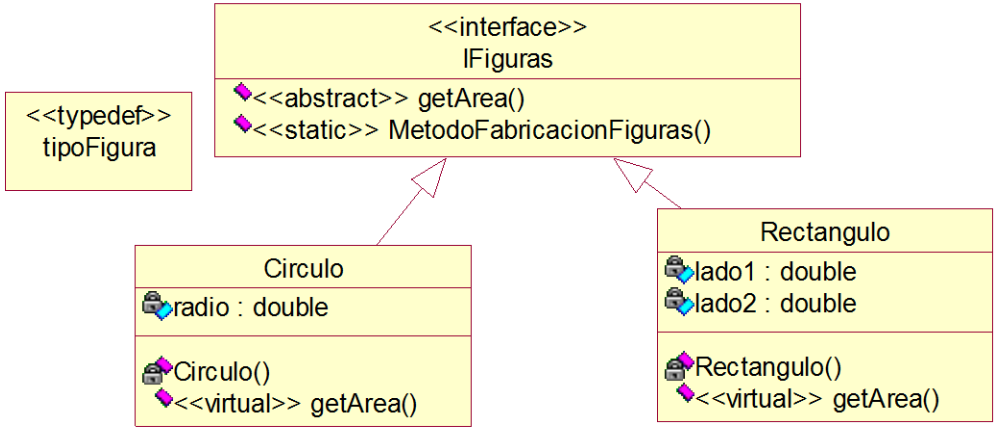





```
IFiguras* IFiguras::MetodoFabricacionFiguras(tipoFigura elTipo,
                                             double param1,double param2 = 0)
{
          if (elTipo == CIRCULO) return new Circulo(param1);
          else if(elTipo == RECTANGULO) return new Rectangulo(param1,param2);
          else return 0;
}
////////////////////////////////////////////////////////////////////////////
void CAreasFiguraDlg::OnCalcular()
{
          UpdateData(TRUE);
          IFiguras *pFigura= IFiguras::MetodoFabricacionFiguras(
                    this->m_Figura == true ? CIRCULO : RECTANGULO,
                    this->m_Param1,this->m_Param2);

          this->m_Area = pFigura->getArea();
          delete pFigura;
          UpdateData(FALSE);
}
```

Ejemplos VP






```
#ifndef _AREAS_FIGURA_INC_
#define _AREAS_FIGURA_INC_
typedef enum tipoFig {CIRCULO, RECTANGULO}
tipoFigura;
class IFiguras
{
public:
 virtual double getArea() = 0;
 static IFiguras* MetodoFabricacionFiguras
          (tipoFigura, double, double);
};
class Circulo: public IFiguras
{
 double radio;
 friend class IFiguras;
 Circulo(double param1):radio(param1) {}
public:
 virtual double getArea()
          {return (3.1416*radio*radio);}
};
```

```
class Rectangulo: public IFiguras
{
 double lado1;
 double lado2;
 friend class IFiguras;
 Rectangulo(double param1, double param2):
          lado1(param1),lado2(param2) {}
 public:
 virtual double getArea()
          {return (lado1*lado2);}
};
#endif
```

Variaciones Protegidas(2/2)

- Otra aplicación de Variaciones Protegidas está en los intérpretes de líneas de comando.
  - P.ej: Matlab

- Principio de sustitución de Liskov:
  - "*El software que hace referencia a un tipo T debería de trabajar correctamente con cualquier implementación o subclase T que la sustituya*".

Ejemplo: patrón Comando

Ejemplo 6.12

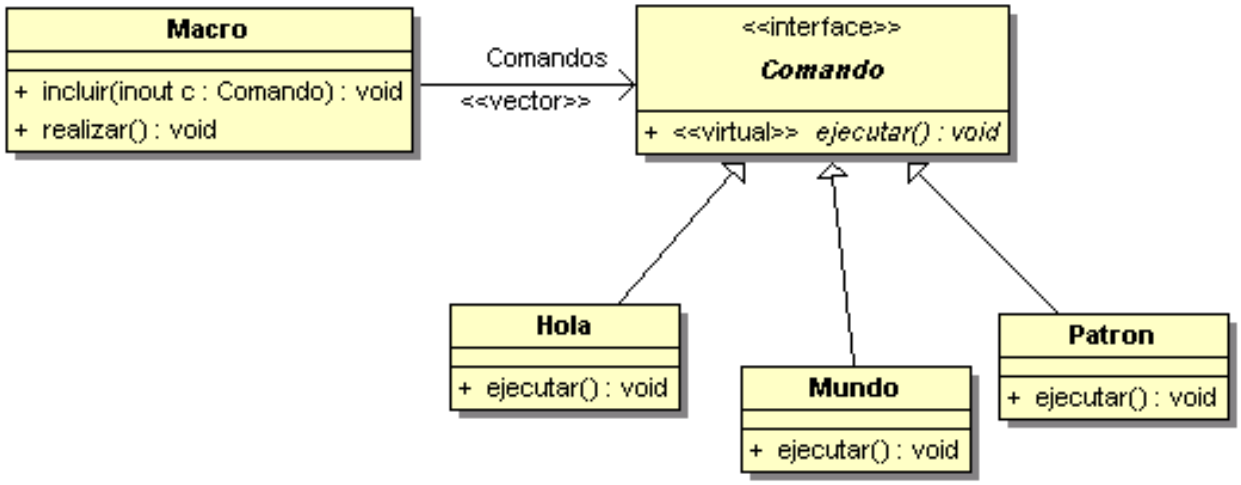
El código entregado corresponde con la implementación del patrón comando, de manera que encapsula un objeto y el cliente lo ve como si fuese una función (muy utilizado en lenguajes script). Se pide:

1) Ingeniería inversa: Diagrama de clases.
2) Ingeniería inversa: Diagrama de secuencias.
3) Resultado de su ejecución en la consola.
4) Indicar los patrones GRASP empleados en este patrón.
5) Diseñar e implementar la clase Saludo, de manera que se despida al añadirse al macro.

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

class Comando
{
public:
  virtual void ejecutar() = 0;
};

class Hola : public Comando
{
public:
  void ejecutar() { cout << "Hola "; }
};

class Mundo : public Comando
{
public:
  void ejecutar() { cout << "Mundo! "; }
};

class Patron : public Comando
{
public:
  void ejecutar() { cout << "Soy el comando patron!"; }
};


class Macro
{
  vector<Comando*> Comandos;
public:
  void incluir(Comando* c) { Comandos.push_back(c); }
  void realizar() {
    for(int i=0;i<Comandos.size();i++)
      Comandos[i]->ejecutar();
  }
};

int main()
{
  Macro macro;
  macro.incluir(new Hola);
  macro.incluir(new Mundo);
  macro.incluir(new Patron);
  macro.realizar();
}
```

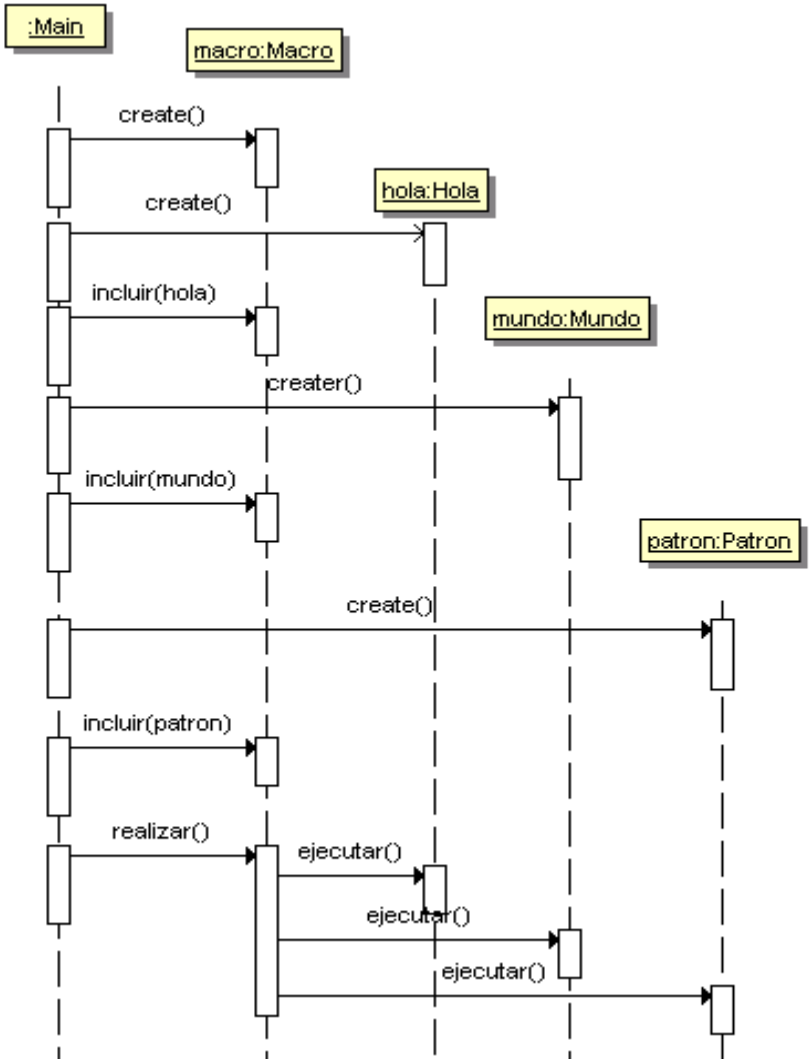
Ejemplo 6.12

1. Ingeniería inversa: diagrama de clases



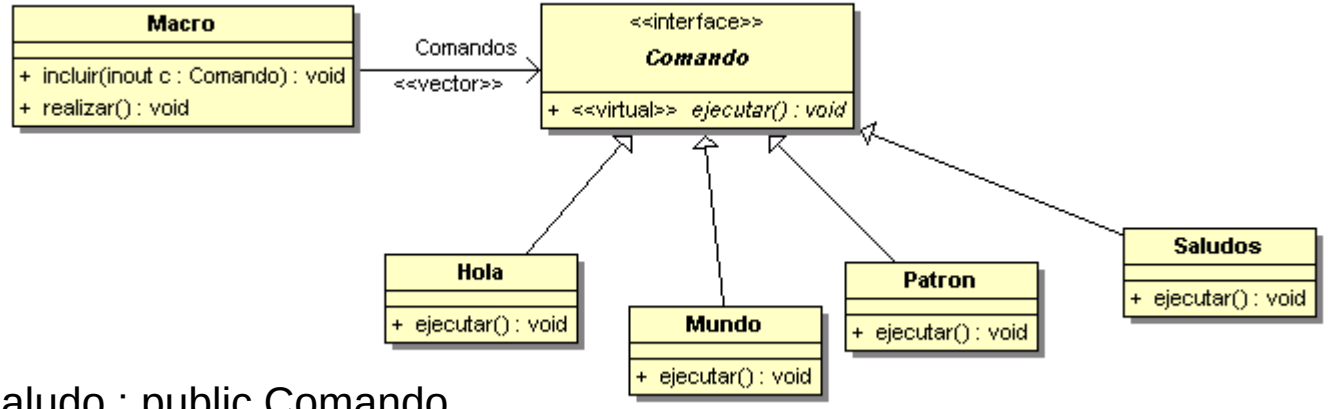
Ejemplo 6.12

2. Ingeniería inversa: diagrama de secuencias



Ejemplo 6.12

3. Resultado consola: Hola Mundo! Soy el comando patron!
4. Patrones: El patrón Comando emplea Variaciones Protegidas (GRASP), de forma que el cliente no ve las modificaciones que está realizando el servidor.
5. Diseño e implementación:




```
class Saludo : public Comando
{
public:
  void ejecutar() { cout << " Un saludo. "; }
};
```

Variaciones Protegidas(2/2)

- "*No hable con Extraños*" o Ley de Demeter. Solo se puede mandar mensajes a:
  1. A él mismo (objeto *this*).
  2. A un parámetro de un servicio propio (visibilidad de parámetro).
  3. A un atributo de él (visibilidad de atributo).
  4. A una colección de él (visibilidad de atributo).
  5. A un objeto creado en un método propio (visibilidad local).

Examen final de julio 2014

Los clasificadores son parte esencial de muchos programas de Ingeniería. Normalmente, hay un conjunto de muestras de entrenamiento, las cuales tienen definido tanto el vector de características como la etiqueta que se asociada a la clase que le corresponde. Hay muchos tipos de clasificadores: Bayes, redes neuronales, lógica borrosa,… De los clasificadores más simples están los denominados kNN (*k-Nearest Neighbours*). En esta primera versión se va a implementar un clasificador de tipo kNN: ante una nueva muestra, ésta queda clasificada con la etiqueta de la muestra de entrenamiento con menor distancia Euclídea entre vectores. Se pide:

1. Ingeniería Inversa de las clases *Muestra* y *BaseEntrenamiento* (3 puntos).
2. Diagrama de clase de diseño DCD en UML de las clases *Clasificador*, *kVecinos* y *Factoria*. Indique los patrones empleados (3 puntos).
3. Implementación de estas últimas clases en C++ (4 puntos).

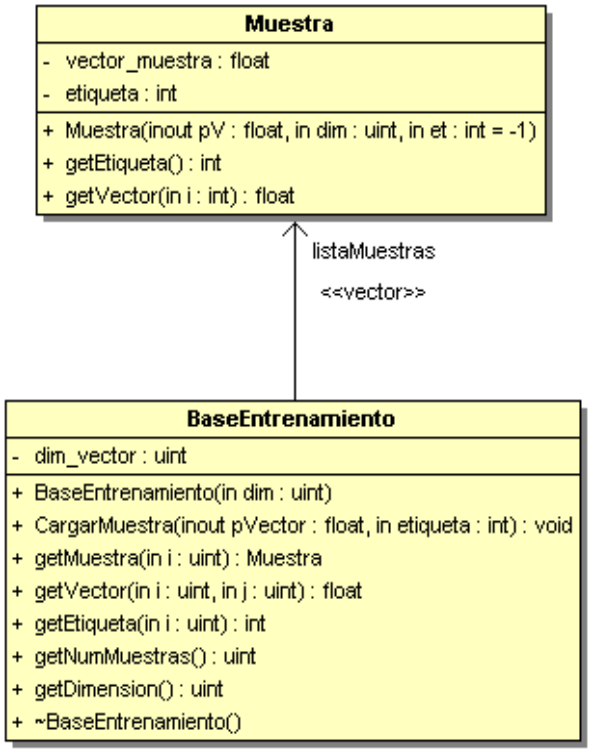
Examen final de julio 2014

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

class Muestra {
  vector<float> vector_muestra;
  int etiqueta;
  public:
  Muestra(float *pV,unsigned dim,int et=-1): etiqueta(et){
            for(unsigned i=0;i<dim;i++)
             vector_muestra.push_back(pV[i]);
            }
  int getEtiqueta(){return etiqueta;}
  float getVector(int i) {return vector_muestra[i];}
};

class BaseEntrenamiento{
  vector<Muestra *> listaMuestras;
  unsigned dim_vector;
  public:
  BaseEntrenamiento(unsigned dim):dim_vector(dim){}
  void CargarMuestra(float *pVector,int etiqueta){
      listaMuestras.push_back(new
      Muestra(pVector,dim_vector,etiqueta));
  }
  Muestra*getMuestra(unsigned i) {return listaMuestras[i];}
   float getVector(int i, int j){return listaMuestras[i]->getVector(j);}
   int getEtiqueta(unsigned i) {return  listaMuestras[i]->getEtiqueta();}
  unsigned getNumMuestras() {return listaMuestras.size();}
  unsigned getDimension() {return dim_vector;}
  ~BaseEntrenamiento() {
    for(unsigned i=0; i<listaMuestras.size();i++)
            delete listaMuestras[i];
     }
};
```

```
typedef enum{kNN,Bayes,RandomForest}
tipoClasificador;
class Clasificador{…};
class kVecinos{…};
class Factoria {…};

int main() {
 unsigned dim = 2;
 unsigned numMuestras = 5;
 float vectores_muestra[5][2] = {
{1.0f,1.0f},{0.0f,0.0f},
            {2.0f,2.0f}, {2.0f,3.0f}, {3.0f,2.0f} };
 int etiquetas_muestra[] = {1,1,2,2,2};
 BaseEntrenamiento laBase(dim);

 for(unsigned i=0;i<numMuestras;i++)
      laBase.CargarMuestra
      (vectores_muestra[i],etiquetas_muestra[i]);

 float vector_nuevo[]={1.0,2.0};
 Muestra laMuestra(vector_nuevo,dim);

 // Clasificar la muestra
 Factoria laFactoria = Factoria::getInstancia();
 Clasificador *pClasificador =
      laFactoria.crearClasificador(kNN,&laBase);
 cout <<"La muestra con vector: "
 << laMuestra.getVector(0)<<" "
 << laMuestra.getVector(1) << " tiene la etiqueta: "
 << pClasificador->getEtiqueta(&laMuestra) << endl;

return 0;
}
```

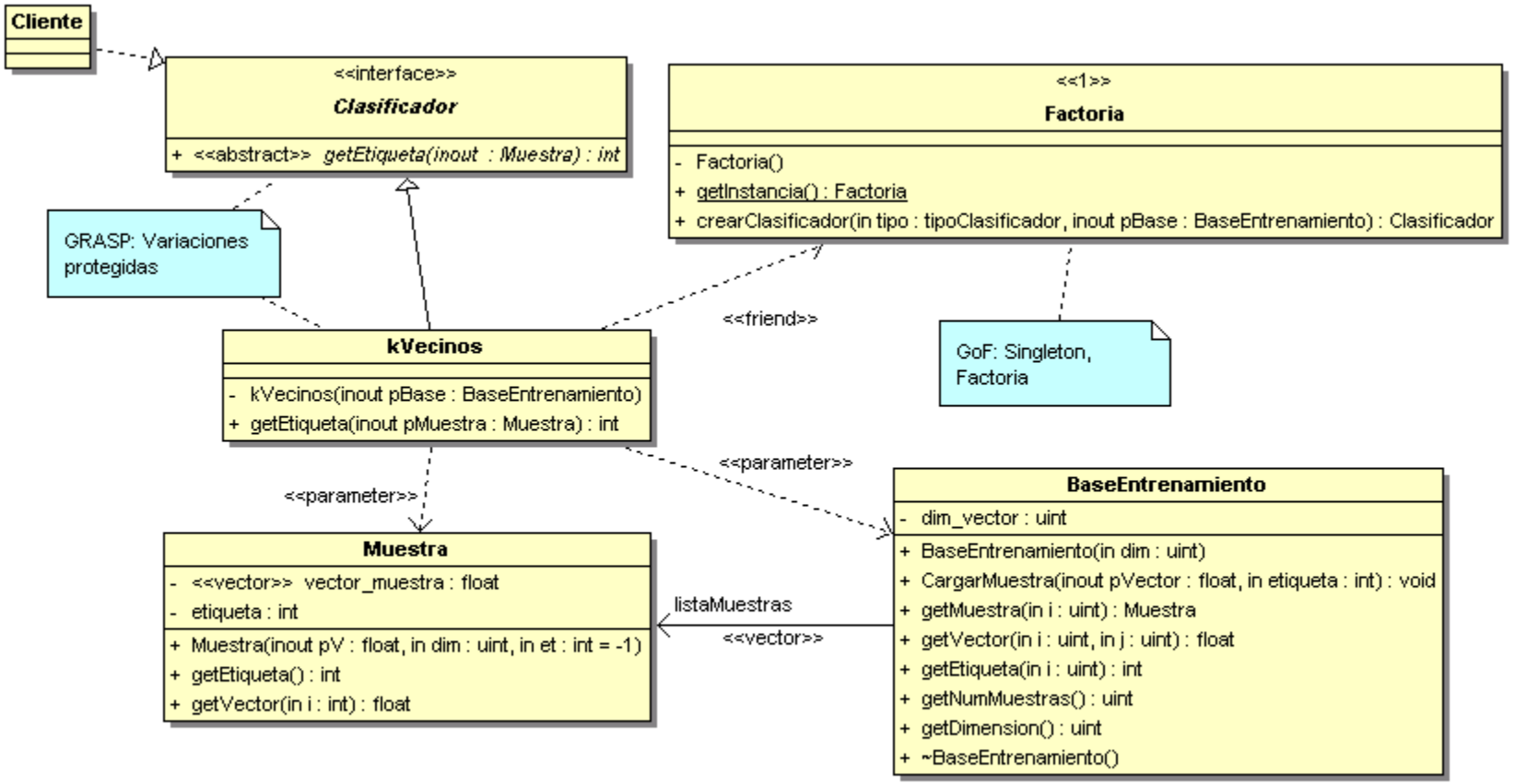
Examen final de julio 2014

1. Ingeniería Inversa de las clases *Muestra* y *BaseEntrenamiento* (3 puntos).



Examen final de julio 2014

2. Diagrama de clase de diseño DCD en UML de las clases *Clasificador*, *kVecinos* y *Factoria*. Indique los patrones empleados (3 puntos).



Examen final de julio 2014

3.Implementación de estas últimas clases en C++ (4 puntos).

```
class Clasificador{
public:
virtual int getEtiqueta(Muestra *) = 0;
};
float distancia_muestras(Muestra *pM1, Muestra *pM2,
                         unsigned dim){
float resultado=0.0f;
for(unsigned i=0; i< dim;i++)
   resultado += (pM1->getVector(i)-pM2->getVector(i))*
                (pM1>getVector(i)-pM2->getVector(i));
return resultado;
}
class kVecinos: public Clasificador {
 friend class Factoria;
 BaseEntrenamiento *pBaseDatos;
 kVecinos(BaseEntrenamiento *pBase):pBaseDatos(pBase) {}
public:
 virtual int getEtiqueta(Muestra *pMuestra) {
  int etiqueta = -1;
  unsigned numMuestras = pBaseDatos->getNumMuestras();
  unsigned dim = pBaseDatos->getDimension();
  float min_distancia = 1e10;
  for(unsigned i=0;i<numMuestras;i++){
     float dist = distancia_muestras
     (pMuestra,pBaseDatos->getMuestra(i),dim);
     if(dist < min_distancia){
        min_distancia = dist;
        etiqueta= pBaseDatos->getEtiqueta(i);
     }
  }
 return etiqueta;
 }
};
```

```
class Factoria
{
Factoria(){}
public:
static Factoria& getInstancia() {
            static Factoria unicaInstancia;
            return unicaInstancia;
}
Clasificador* crearClasificador(tipoClasificador tipo,
                        BaseEntrenamiento *pBase ){
 if(tipo == kNN) return new kVecinos(pBase);
 else return NULL;
}

};
```