UML dinámico y de implementación

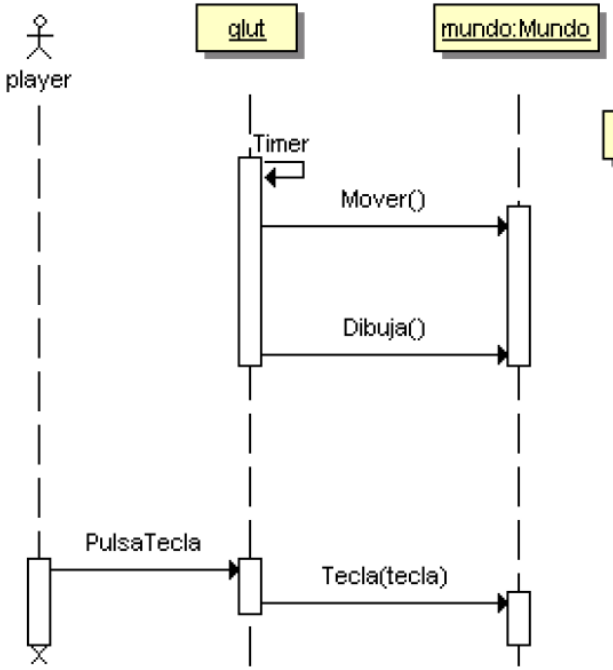
carlos.platero@upm.es (C-305)

UML dinámico

- Vistas de UML
  - Estructural, Dinámico, Implementación
- Dinámico o de evolución temporal
  - Diagramas de Interacción (Secuencia, Colaboración),
  - Diagramas de Estado,
  - Diagramas de Actividades.
- Diagramas de Interacción
  - Interacción entre los objetos por medio de mensajes
  - POO: Asignación de responsabilidades
  - Artefactos para aplicar patrones, estilos y principios
  - Secuencial y Colaboración (complementarios)

Diagrama de secuencia(1/2)

- Diagramas de secuencia
  - Vertical -> tiempo, Horizontal->objetos
  - Líneas de vida, mensajes, actividades.



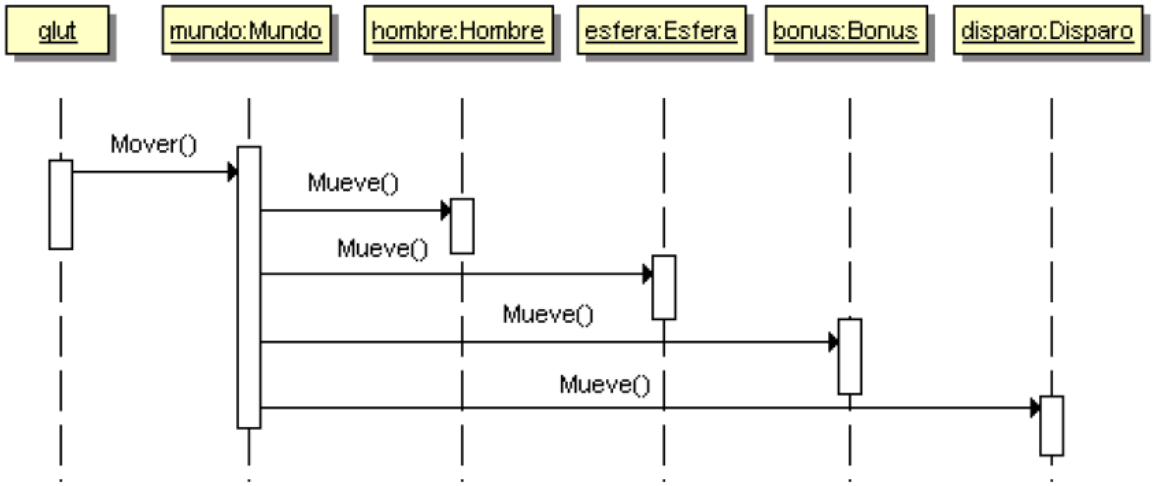
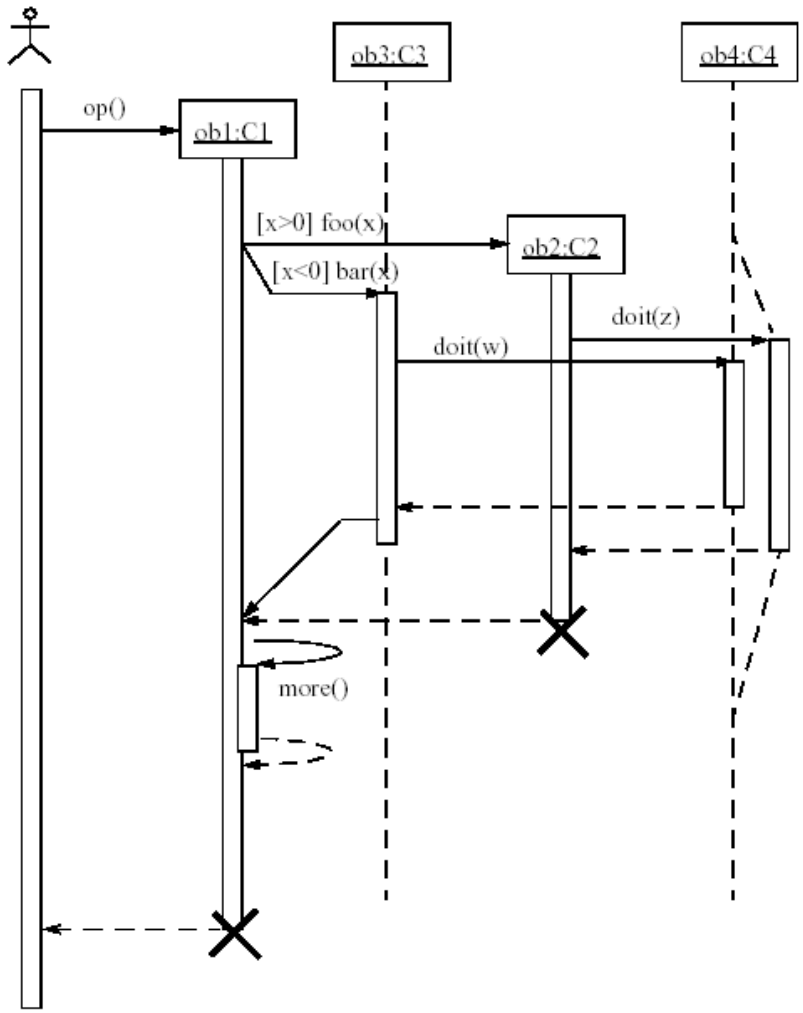


Diagrama de secuencia(2/2)

- UML ofrece notación para:
  - Mensajes condicionales
  - Iteraciones
  - Mensajes sobre si mismos

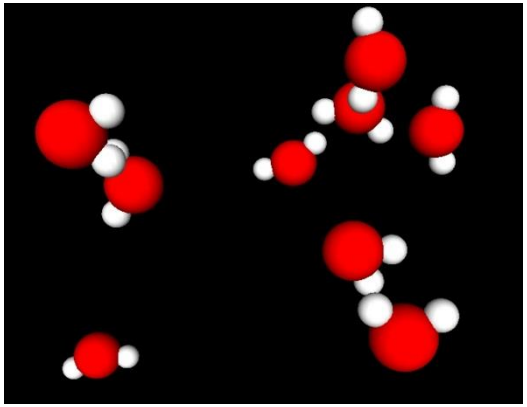


Problema vapor de agua

Se pretende desarrollar un programa de simulación de un vapor de agua, para el que se ha desarrollado ya la siguiente clase, que funciona correctamente y no es necesario modificar:

```
class Atomo
{
public:
 Atomo(int num);
 virtual ~Atomo();
 void Enlaza(Atomo* a);
 void CalculaPosicion();
 void Dibuja();
protected:
          float x;
          float y;
          float z;
          int numero_atomico;
          Atomo* enlace;
};
```

Ejemplo del resultado final



Problema vapor de agua

El código para dibujar una molécula de agua utilizando la clase Atomo sería el siguiente:

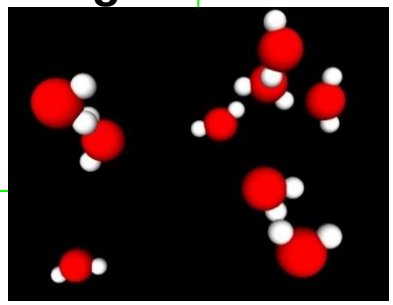
```
void main()
{
        Atomo hidrogeno1(1);
        Atomo hidrogeno2(1);
        Atomo oxigeno(16);

        hidrogeno1.Enlaza(&oxigeno);
        hidrogeno2.Enlaza(&oxigeno);
        oxigeno.CalculaPosicion();
        hidrogeno1.CalculaPosicion();
        hidrogeno2.CalculaPosicion();

        hidrogeno1.Dibuja();
        hidrogeno2.Dibuja();
        oxigeno.Dibuja();
}
```

Donde es importante el orden de cálculo de las posiciones, es decir, primero se calcula la posición del oxigeno, y después la de los átomos de hidrogeno, que dependen del átomo de oxigeno.

Problema vapor de agua

Cuando el programa este completado, el código del main() debe quedar como:

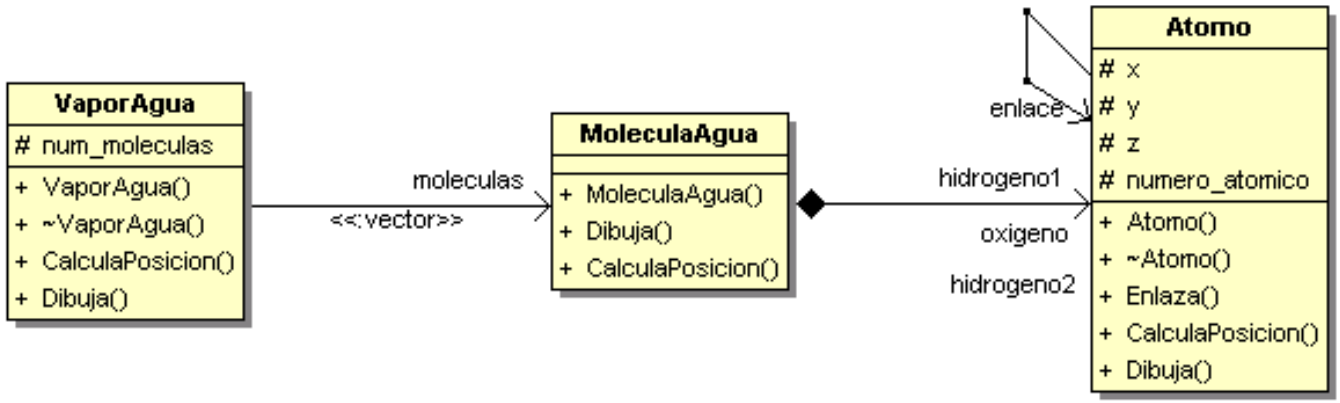
```
void main()
{
        VaporAgua vapor(30);      //30 moleculas de agua
        vapor.CalculaPosicion();
        vapor.Dibuja();
}
```



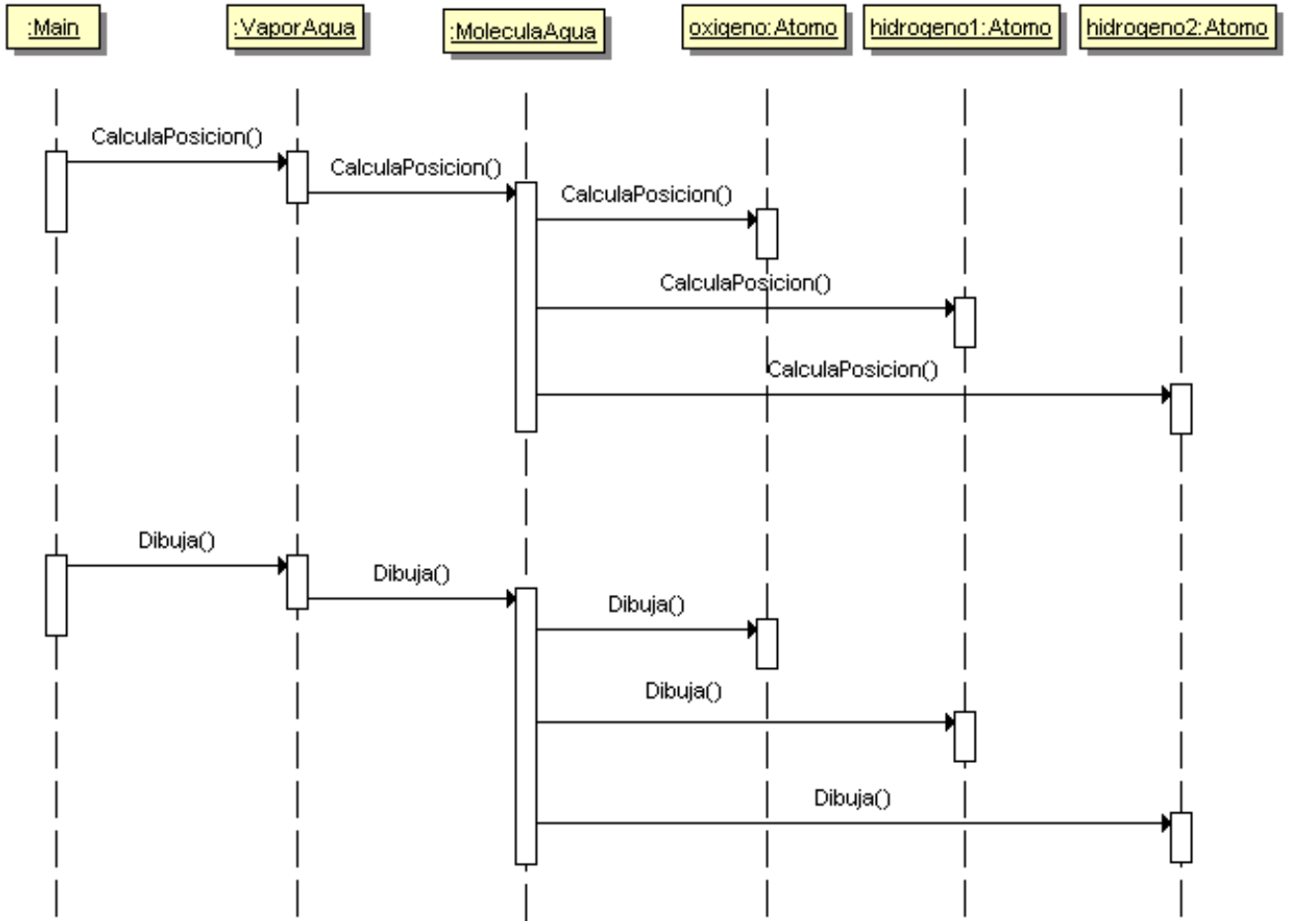
- **Se pide:**

1. Diagrama de Clases de Diseño (DCD de la solución), que incluya orientación a objetos para cada molécula de agua.
2. Un diagrama de secuencias del cálculo de la posición y el dibujo del vapor, a partir de la función main().
3. Implementación de la solución en C++

Problema vapor de agua



Problema vapor de agua



Problema vapor de agua

```
class MoleculaAgua
{
public:
        MoleculaAgua();
        void Dibuja();
        void CalculaPosicion();
protected:
        Atomo hidrogeno1;
        Atomo hidrogeno2;
        Atomo oxigeno;
};
```

```
//MoleculaAgua.cpp
MoleculaAgua::MoleculaAgua():oxigeno(16),
hidrogeno1(1),hidrogeno2(1)
{
        hidrogeno1.Enlaza(&oxigeno);
        hidrogeno2.Enlaza(&oxigeno);
}
void MoleculaAgua::CalculaPosicion()
{
        oxigeno.CalculaPosicion();
        hidrogeno1.CalculaPosicion();
        hidrogeno2.CalculaPosicion();
}
void MoleculaAgua::Dibuja()
{
        hidrogeno1.Dibuja();
        hidrogeno2.Dibuja();
        oxigeno.Dibuja();
}
```

Problema vapor de agua

```
#include <vector>
class VaporAgua
{
public:
        VaporAgua(int num);
        virtual ~VaporAgua();
        void CalculaPosicion();
        void Dibuja();
protected:
        std::vector<MoleculaAgua *>
        moleculas;

};
```

```
VaporAgua::VaporAgua(int num)
{
        for(int i=0;i<num;i++)
            moleculas.push_back(new MoleculaAgua);

}
VaporAgua::~VaporAgua()
{
        for(int i=0;i<moleculas.size();i++)
                delete moleculas[i];

}
void VaporAgua::Dibuja()
{
        for(int i=0;i<moleculas.size();i++)
                moleculas[i]->Dibuja();
}

void VaporAgua::CalculaPosicion()
{
        for(int i=0;i<moleculas.size();i++)
                moleculas[i]->CalculaPosicion();
}
```

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

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

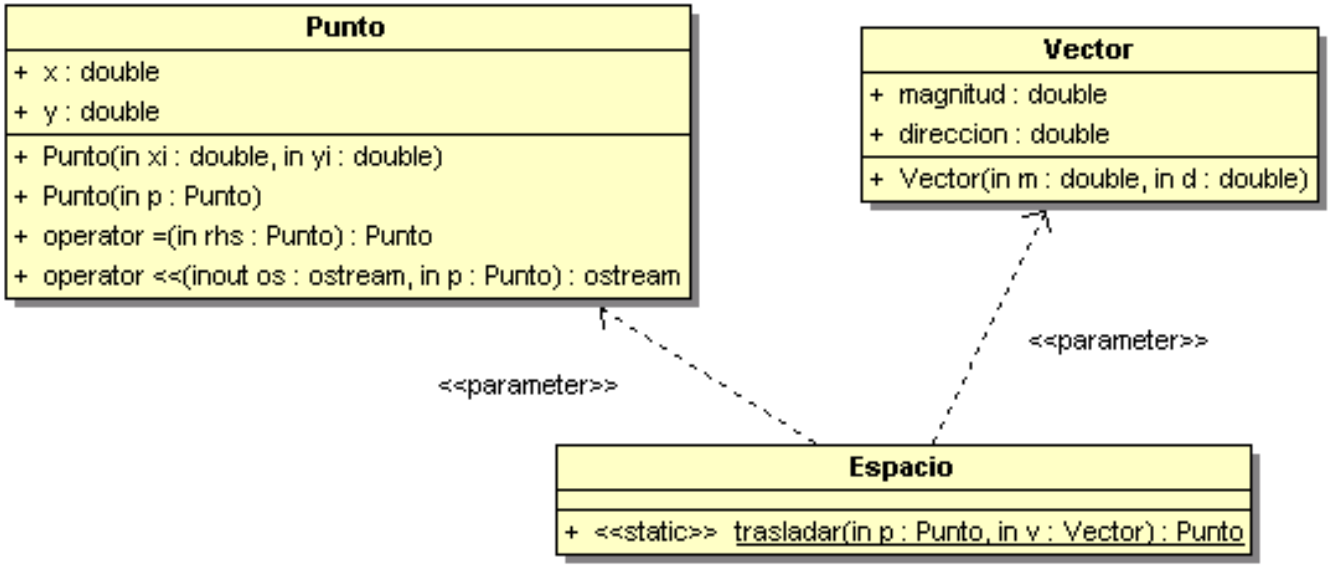
Empléese sobre el punto p3.

Problema de examen

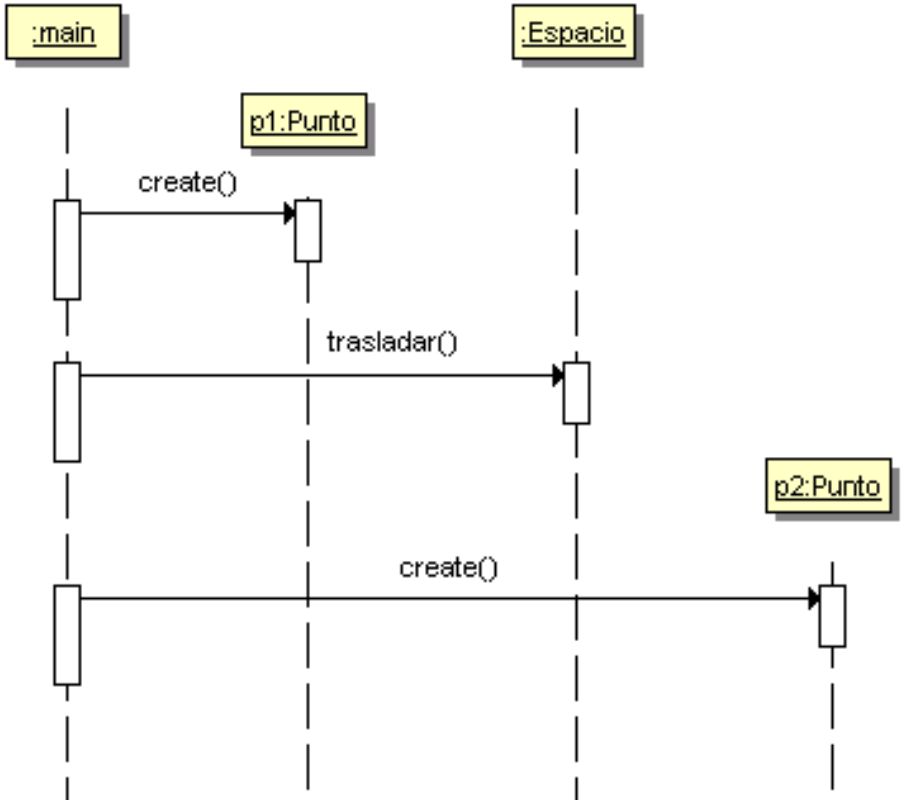
```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

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



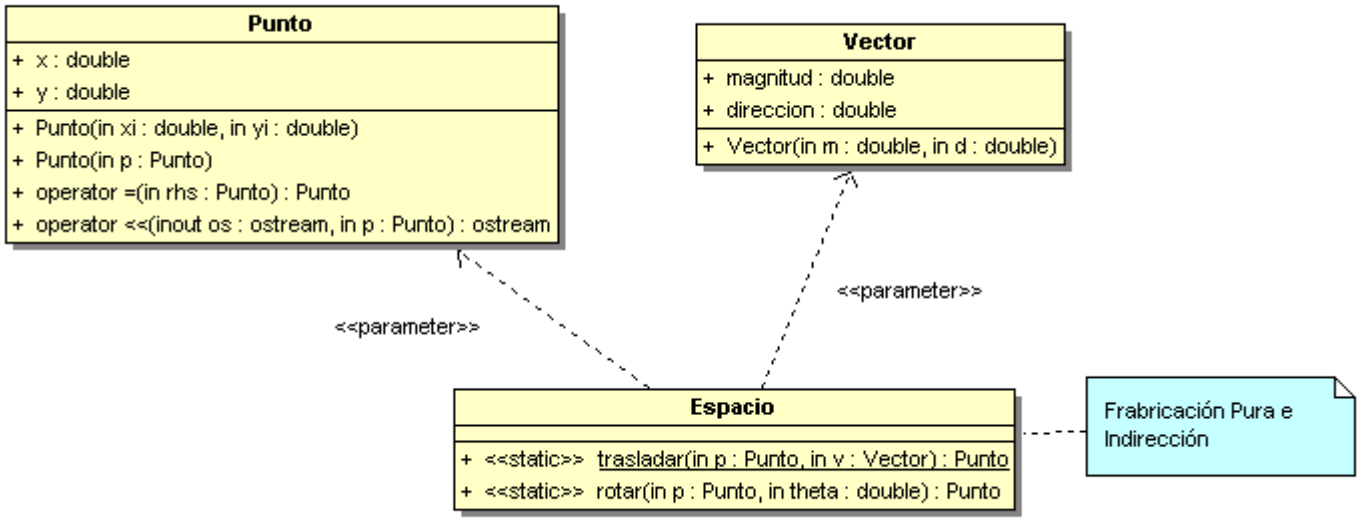
Problema de examen



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.



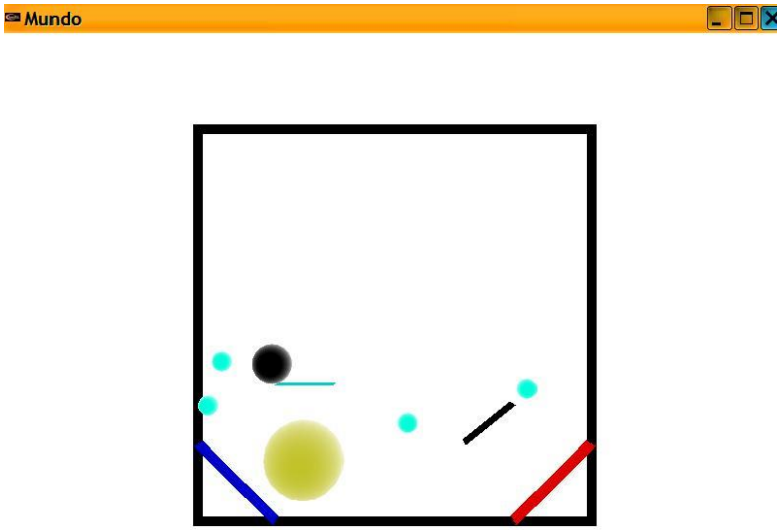
Problema de examen

```
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;
}
```

Ejercicio

Se desea hacer una aplicación que simule la interacción entre un número indefinido de esferas dentro de una caja cerrada y que tiene barreras:

1. Principales características: jerarquía a dos niveles.
2. Modelo del dominio.
3. Vista de gestión
4. Diagrama de Clases de Diseño indicando los patrones empleados.
5. Diagrama de secuencia del servicio **Mueve(t: float) : void**.
6. Implementación en C++ de la solución.



Ejercicio

Se desea hacer una aplicación que simule la interacción entre un número indefinido de esferas dentro de una caja cerrada y que tiene barreras:

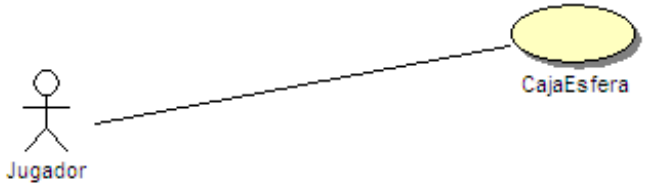
I. Principales características: jerarquía a dos niveles.

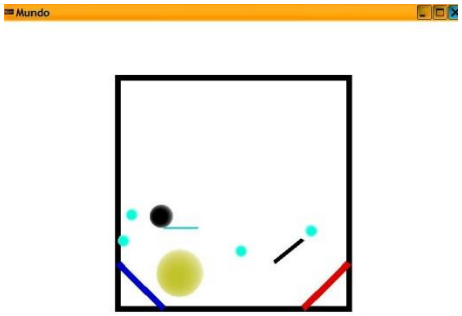
1. El sistema debe de simular la interacción entre una lista de esferas contra las paredes y las barreras.

1.1. Las esferas tienen velocidades y aceleraciones aleatorias

1.1 El sistema debe determinar la colisión entre cualquier esfera y las paredes o barreras, cambiando la velocidad de las esferas que chocan.

1.2 El sistema debe determinar el choque entre las esferas

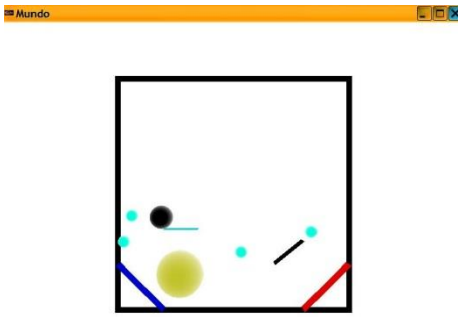


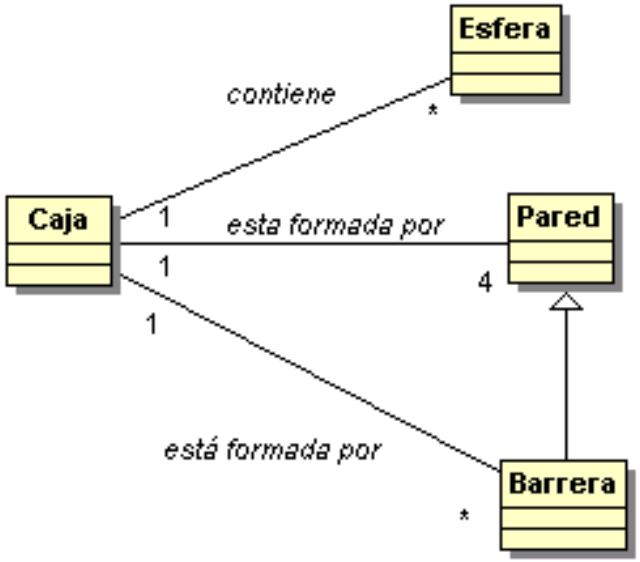


1.0 Se pinta la caja, las barreras y las esferas
2.0 Se mueven las esferas aleatoriamente
3.0 Si chocan contra las barreras, paredes u otras esferas cambian el movimiento

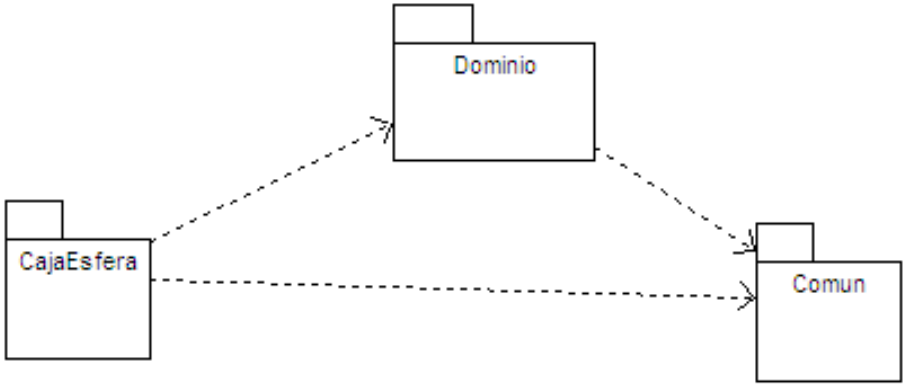
Ejercicio

- Glosario: Caja, Pared, Barrera, Esfera

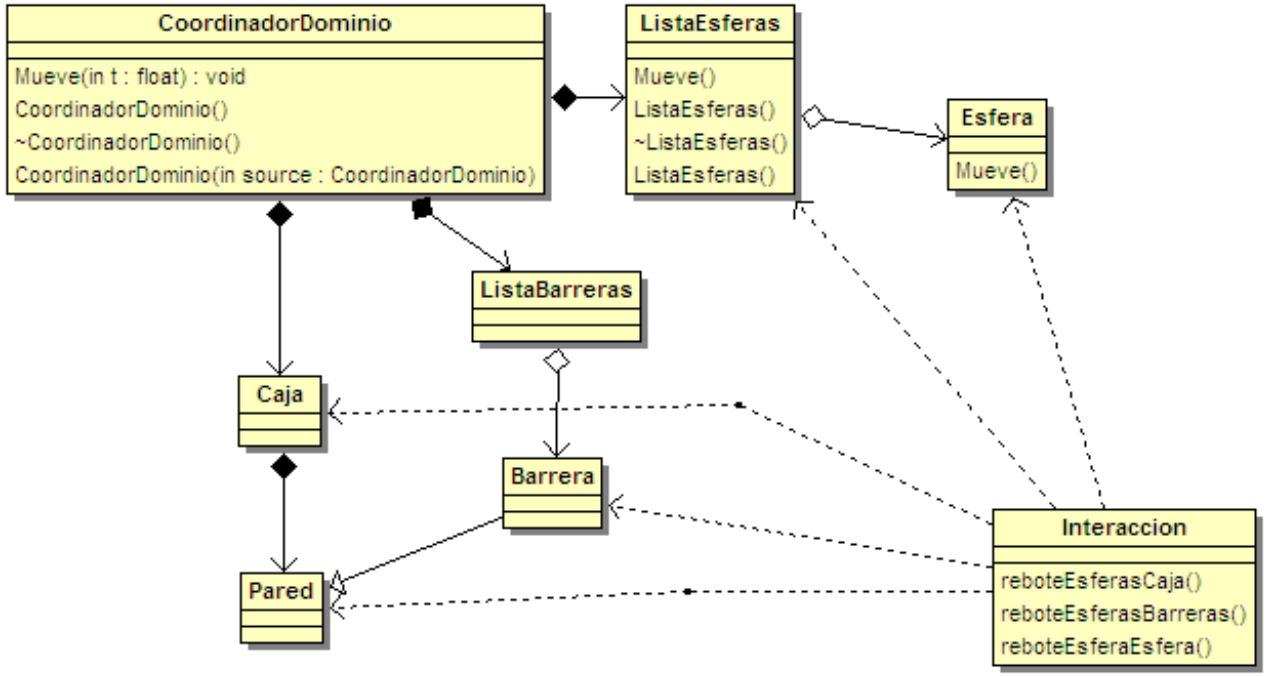


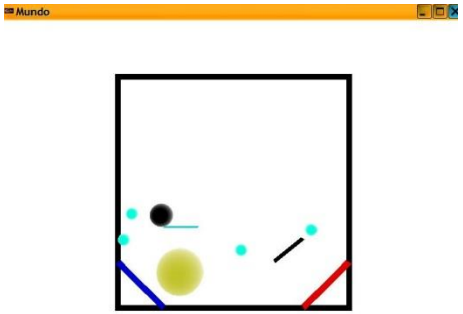


Ejercicio

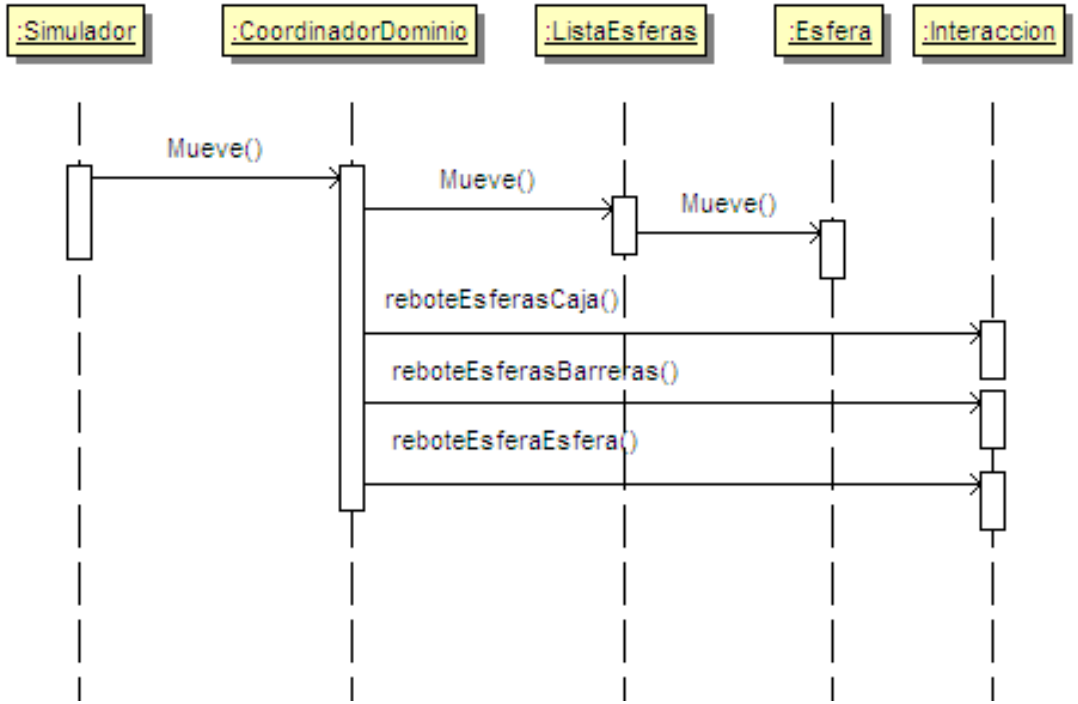


Ejercicio


CoordinadorDominio
Mueve(in t : float) : void
CoordinadorDominio()
~CoordinadorDominio()
CoordinadorDominio(in source : CoordinadorDominio)
ListaEsferas
Mueve()
ListaEsferas()
~ListaEsferas()
ListaEsferas()
Esfera
Mueve()
ListaBarreras
Caja
Barrera
Pared
Interaccion
reboteEsferasCaja()
reboteEsferasBarreras()
reboteEsferaEsfera()


Mundo

Ejercicio


:Simulador
:CoordinadorDominio
:ListaEsferas
:Esfera
:Interaccion
Mueve()
Mueve()
Mueve()
reboteEsferasCaja()
reboteEsferasBarreras()
reboteEsferaEsfera()

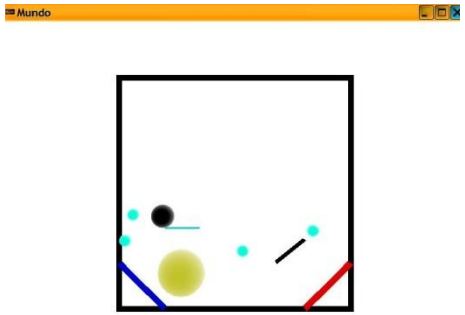
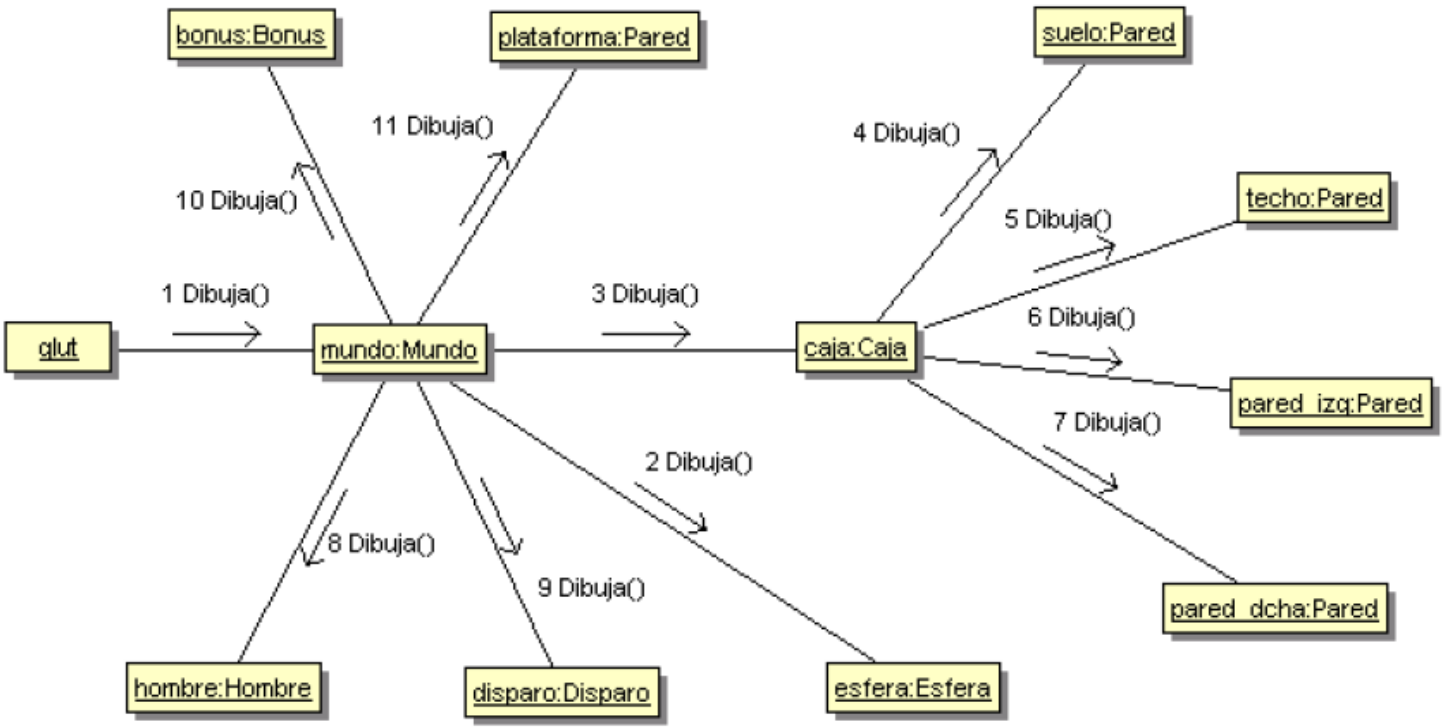

Mundo

Diagrama de colaboración

- Secuencia->tiempo
- Colaboración-> relaciones de roles
- Complementarios

- Diagramas de colaboración
  - Modela enlaces entre objetos.
  - Enlace: camino de conexión (navegabilidad y visibilidad)
  - Instancia de asociación
  - Mensaje: expresión, flecha de flujo y número de secuencia
  - Numeración
    - Anidamiento según orden de salida

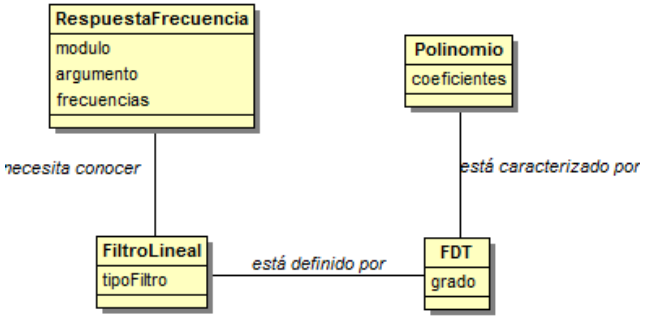
Ejemplo de diagrama de colaboración

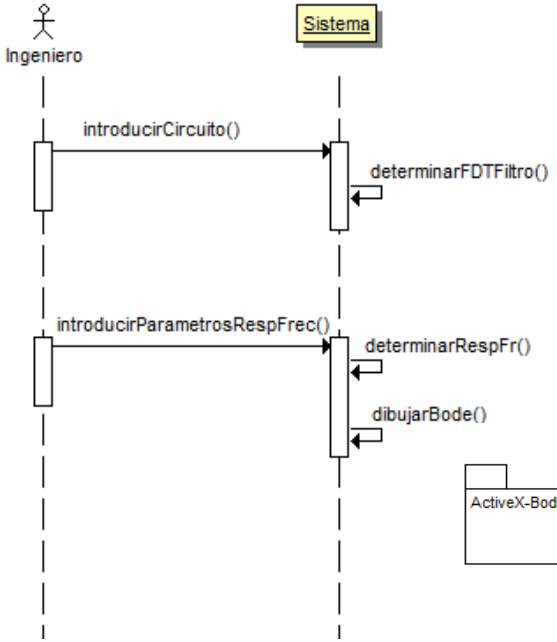


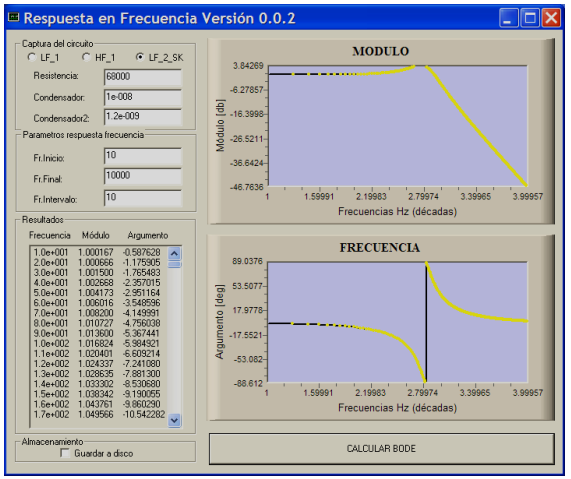
Ejemplo en Respuesta en Frecuencia

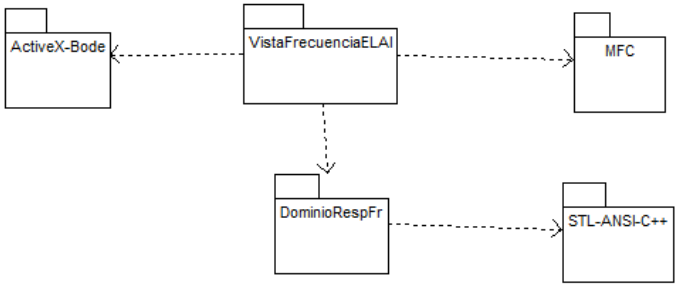
Para el caso de uso *RespuestaFrecuencia* realizar el diagrama de secuencia y colaboración del paquete del dominio.

Partiendo del Modelo del Dominio, DSS, contratos de operación y vista de gestión, visto en el capítulo 3 y 4:



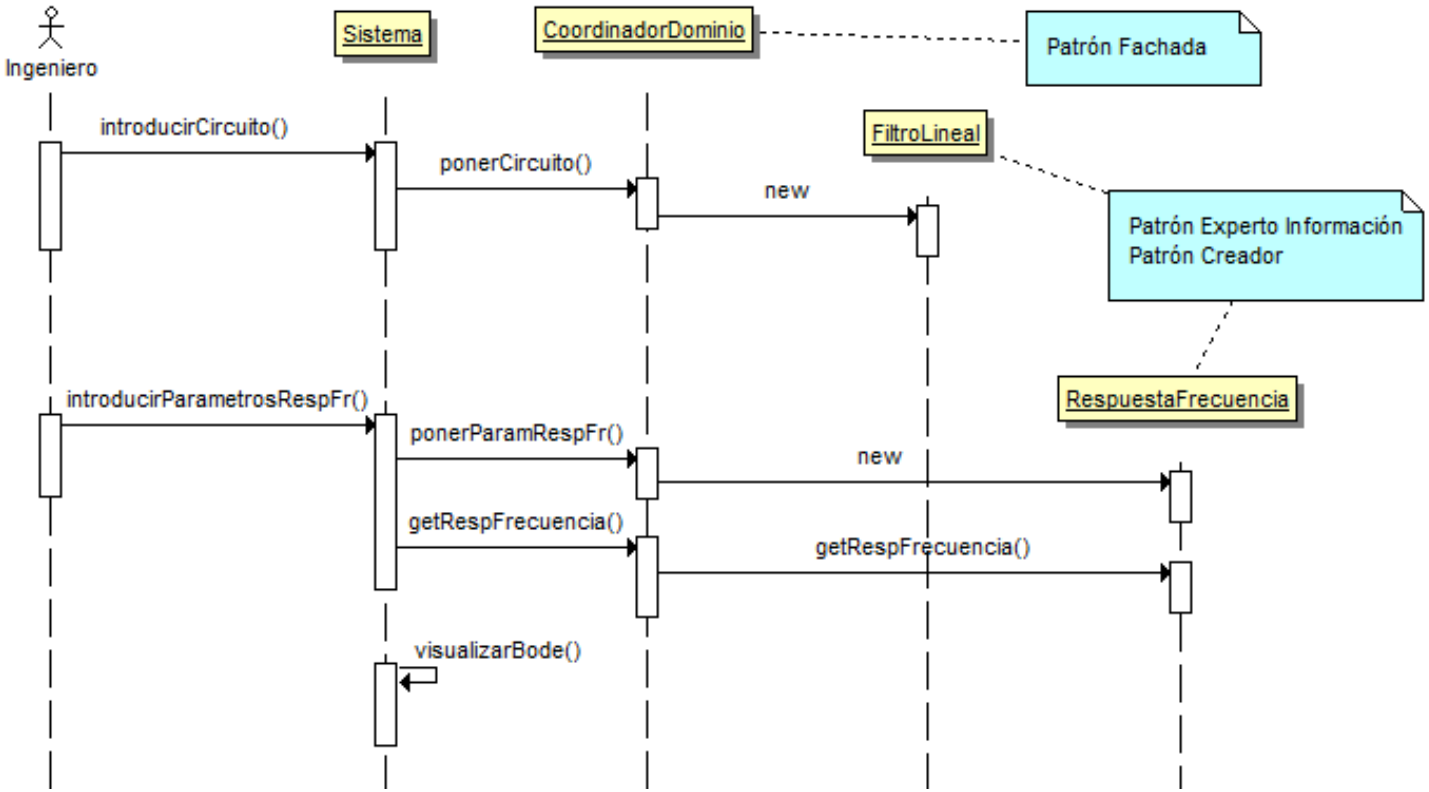






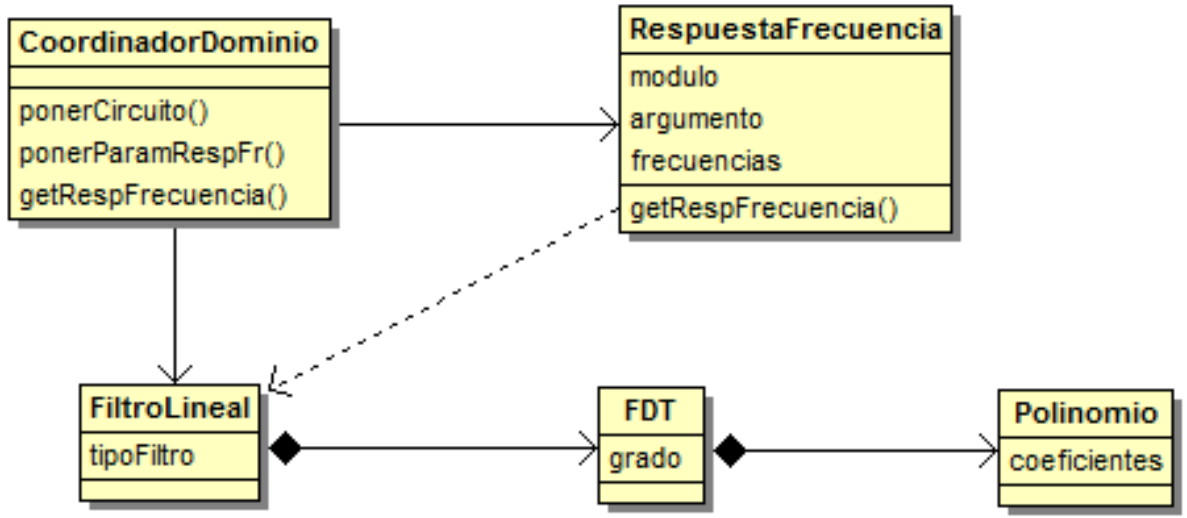
Ejemplo en Respuesta en Frecuencia

Aplicando patrones, estilos y principios de la POO:



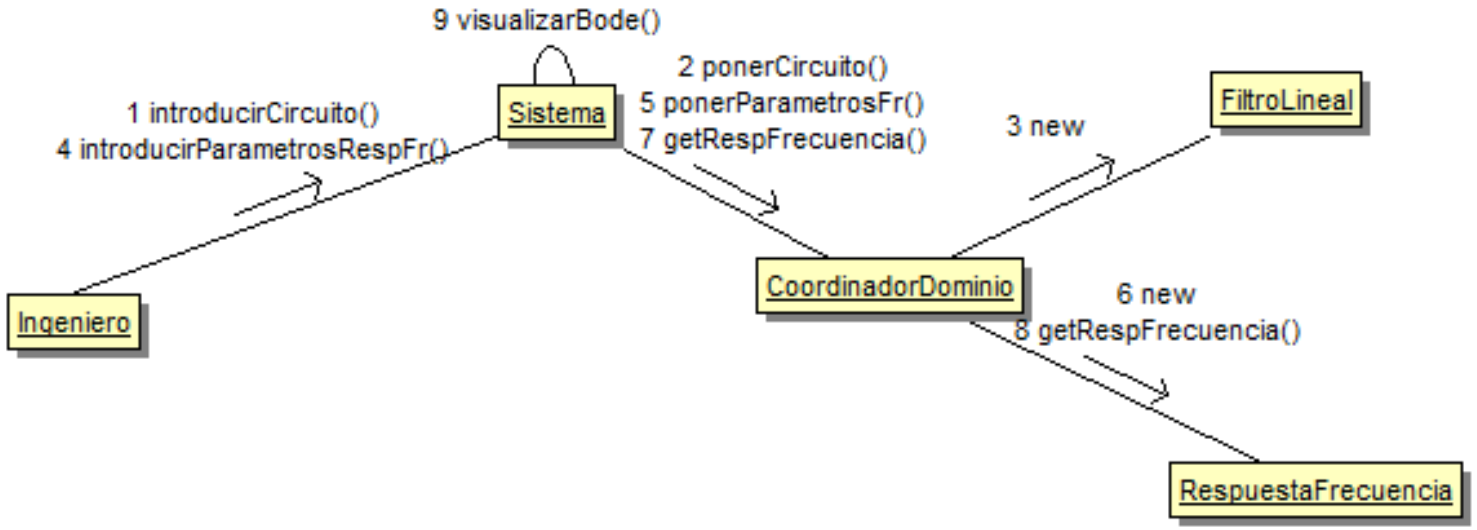
Ejemplo en Respuesta en Frecuencia

Aplicando patrones, estilos y principios de la POO:



Ejemplo en Respuesta en Frecuencia

Aplicando patrones, estilos y principios de la POO:



Diagramas de estado (1/2)

- Modela los posibles estados de un objeto
- Máquina de estado:
  - Evento: ocurrencia significativa
  - Estado: valores de los atributos en un determinado tiempo (UML: Rectángulo con esquinas redondeadas)
  - Transición: relación entre dos estados por un evento (UML: flechas etiquetadas con los eventos)
- Utilidad:
  - Comprobar que los eventos ocurren en orden correcto.
  - Habilitar/deshabilitar elementos según el desarrollo del diagrama de estados.
  - En un dominio con muchos eventos del sistema, la concisión y minuciosidad del diagrama de estado ayuda al diseñador a asegurarse a que no se han omitido ninguno.

Ejemplo del juego Pang

- Realizar una máquina de estado del videojuego del *Pang*

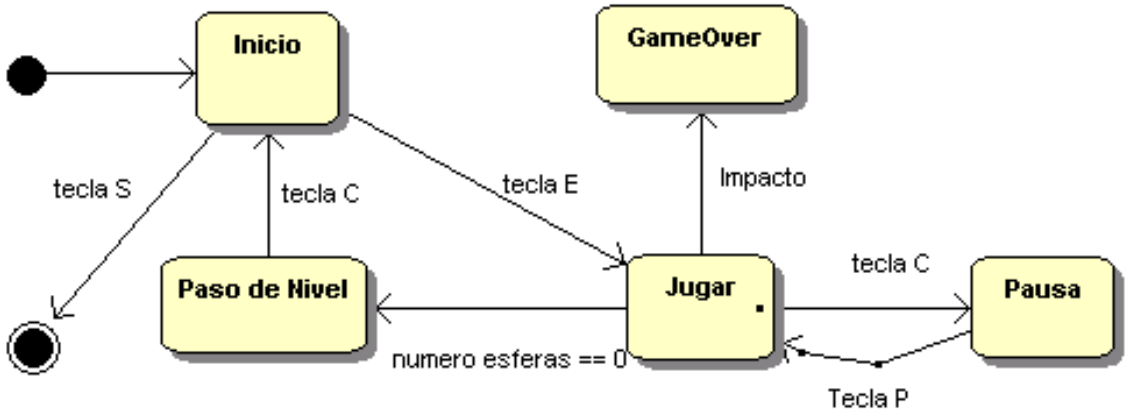
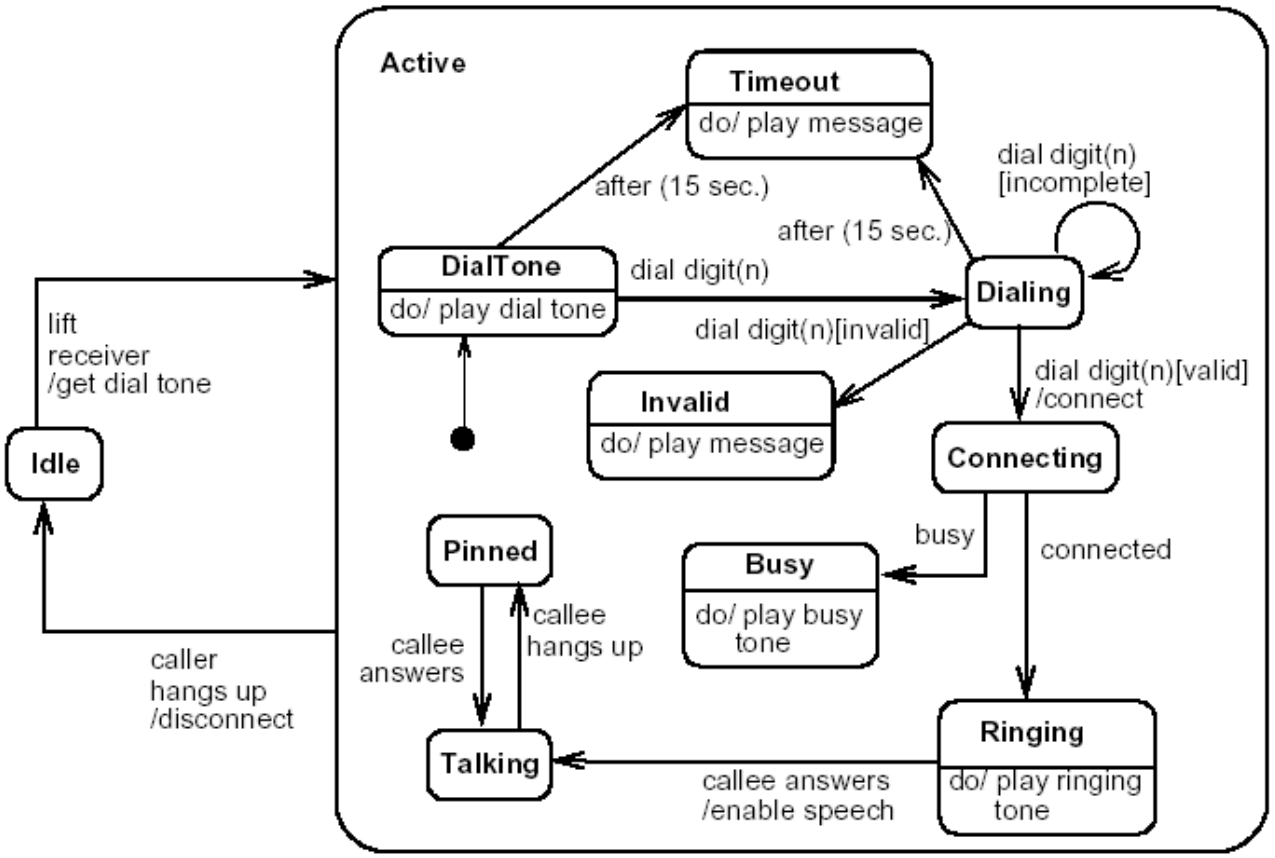


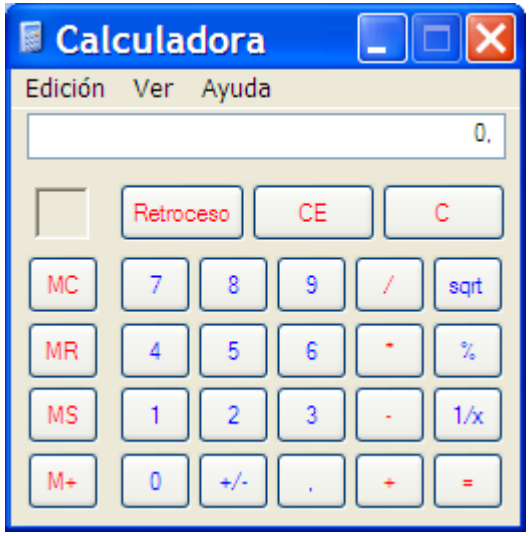
Diagrama de estado(2/2)

- Gestión versus control de procesos, telecomunicaciones.
- Eventos: Externos, internos, tiempo
  - Máquinas de estado: eventos externos y de tiempo.
  - Diagramas de interacción: eventos internos



Problema calculadora

Determinar la máquina de estado de una calculadora



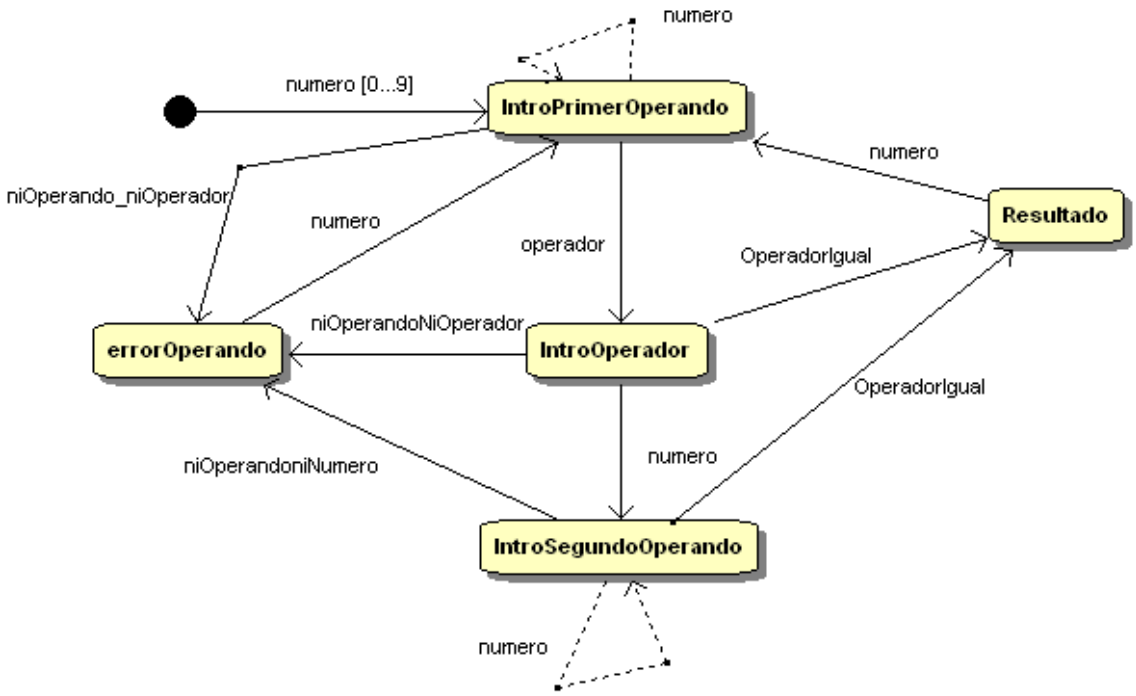
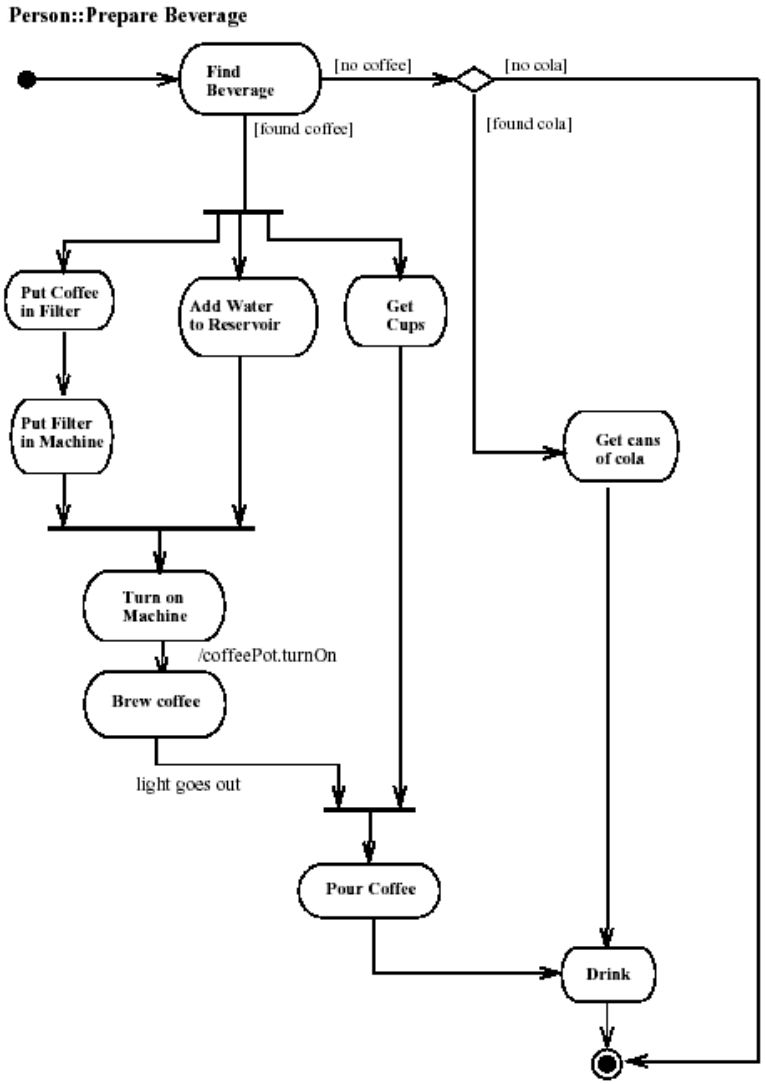


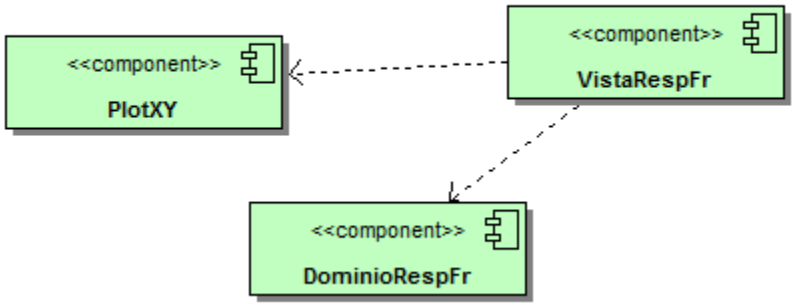
Diagrama de actividades

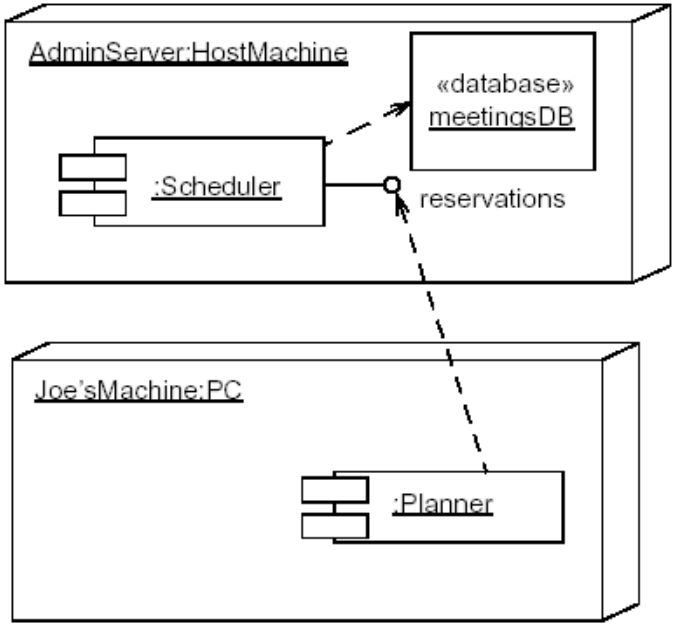
- Visualizar flujos de trabajo
- Un tipo de diagrama de estado
  - Los estados son acciones
  - Las transiciones se disparan automáticamente
- Utilidad: modelar actividades de alto nivel



Vista física

- Modelan la estructura de implementación
  - Diagrama de componentes
    - Componentes: objetos físicos en tiempos de compilación, desarrollo o ejecución (UML: 3 rectángulos).
    - Utilidad: dependencias y sustituciones
      - Cualquier componente puede ser sustituido por otro que tenga el mismo interfaz
  - Diagrama de despliegue
    - Nodos: objetos físicos en tiempo de ejecución (UML: Paralelepípedo)
    - Utilidad: arquitectura, rendimientos.





Cuestiones

1. Diferencias entre el diagrama de secuencia con el de colaboración.
2. Obtener el diagrama de secuencias y de colaboración para la aplicación *Juego de dados:* se lanzan dos dados, si la suma de sus caras es siete gana; en caso contrario, pierde.
3. Utilidades de los diagramas de estado.
4. Cuándo se empleará un diagrama de estado y cuándo de interacción.

Problemas

Ejercicio 1

Realizar una aplicación que ordenen de forma creciente los números dados por el usuario.

Ejercicio 2

Diseñar el programa de una maquina expendedora de bebidas, de manera que recibe el producto seleccionado y las monedas que entran en el cajero desde un autómata programable. La aplicación debe de retornar la lista de monedas mínimas a entregar al usuario.

Ejercicio 1

Realizar una aplicación que ordene de forma creciente los números dados por el usuario.

1. Caso de Uso
2. Modelo del dominio y DSS
3. Vista de Gestión.
4. Diagrama de secuencia y diagrama de clases de diseño
5. Implementación en C++

Ejercicio 1

Realizar una aplicación que ordene de forma creciente los números dados por el usuario.

1. Caso de Uso

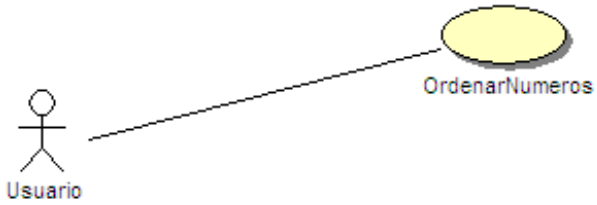
   El caso de uso EBP se llama *"OrdenarNumeros"* y tendrá un curso de éxito como:

   I. Ordenar los números de forma creciente

      I.a. Solicitar al usuario el vector de números a ordenar

      I.b. Ordenar de forma creciente el vector

      I.c. Visualizar los resultados

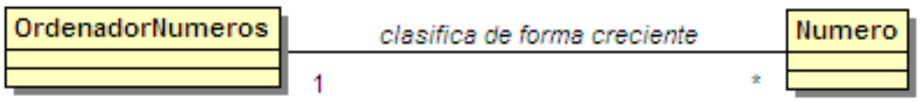


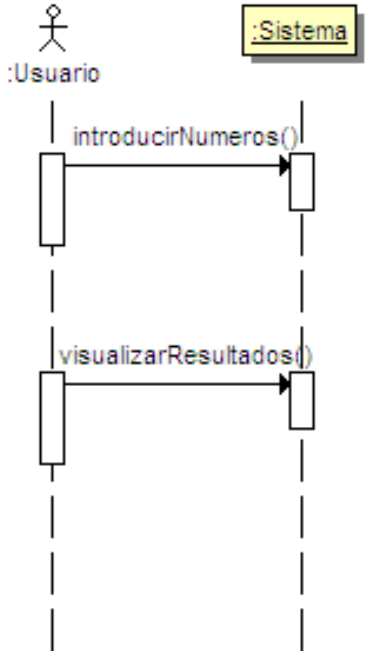
Ejercicio 1

Realizar una aplicación que ordene de forma creciente los números dados por el usuario.

2. Modelo del dominio y DSS

Del análisis de los documentos aparece los conceptos de *OrdenadorNumeros* y *Número* proponiéndose el siguiente Modelo del Dominio y DSS:

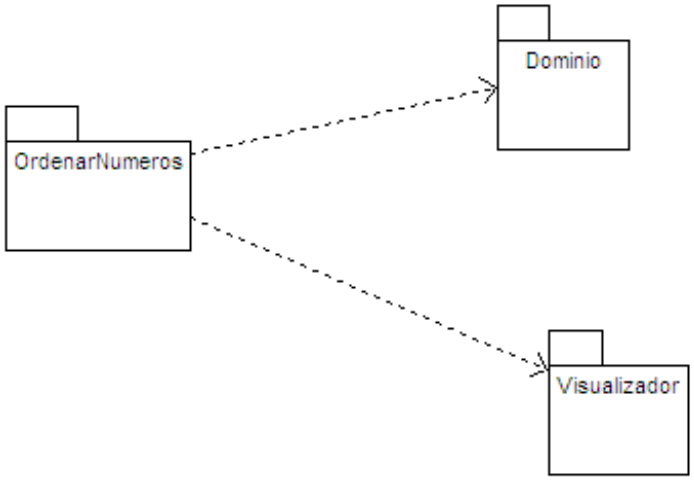




Ejercicio 1

Realizar una aplicación que ordene de forma creciente los números dados por el usuario.

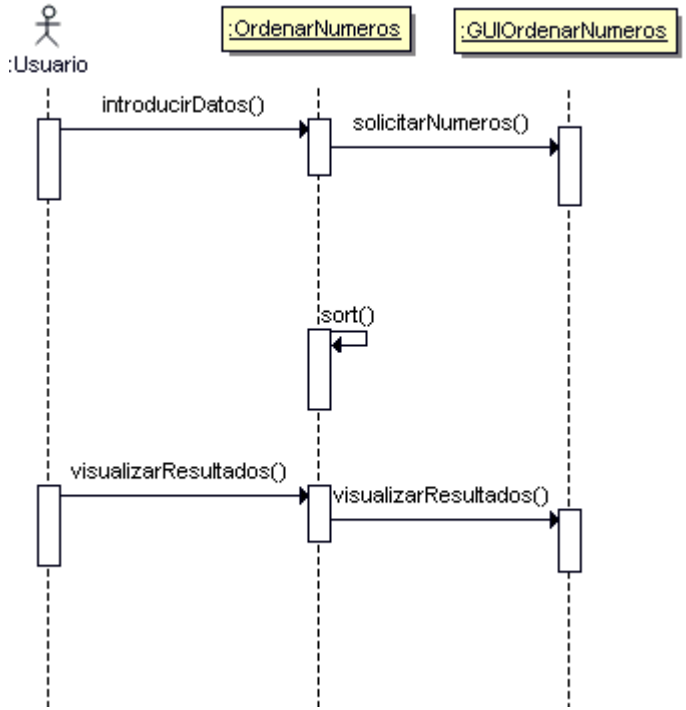
3. Vista de Gestión.

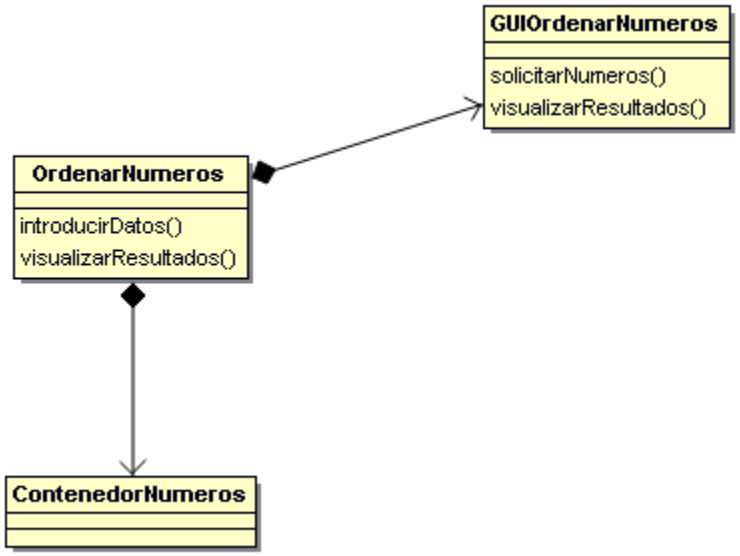


Ejercicio 1

Realizar una aplicación que ordene de forma creciente los números dados por el usuario.

4. Diagrama de secuencia y diagrama de clases de diseño





Ejercicio 1

5. Implementación en C++

Código de test

```
int main()
{
        OrdenarNumeros elOrdenar;
        elOrdenar.introducirDatos();
        elOrdenar.visualizarResultados();
        return 0;
}
```

Ejercicio 1

5. Implementación en C++

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

#include <vector>

class GUIOrdenarNumeros
{
public:
		void visualizarResultados(std::vector<double> &);
		void solicitarNumeros(std::vector<double> &);
};
#endif
```

```
#include "..\..\CABECERAS\VISUALIZADOR\GUIOrdenarNumeros.h"
#include <algorithm>
#include <iostream>
void GUIOrdenarNumeros::solicitarNumeros(std::vector<double> &elVectorNumeros)
{
		bool introducirDatos = true; double valor;
		std::cout<<"Esta aplicacion ordena los valores de forma creciente"<<std::endl;
		std::cout<<"Introducir la lista de numeros y poner cero para salir"<<std::endl;
		while(introducirDatos == true){
				std::cin>>valor;
				if(valor != 0)
						elVectorNumeros.push_back(valor);
				else
						introducirDatos = false;
		}
}
void visualizarDatos(double);
void GUIOrdenarNumeros::visualizarResultados(std::vector<double> &elVectorNumeros)
{
		std::cout<<"Lista ordenada"<<std::endl;
		std::for_each(elVectorNumeros.begin(),elVectorNumeros.end(),visualizarDatos);
}
void visualizarDatos(double valor)
{
		std::cout<<valor<<std::endl;
}
```

Ejercicio 1

5. Implementación en C++

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

#include <vector>
#include <algorithm>
#include "../Visualizador/GUIOrdenarNumeros.h"

class OrdenarNumeros
{
                GUIOrdenarNumeros elVisualizador;
                std::vector<double> elVectorNumeros;
public:
                void introducirDatos();
                void visualizarResultados();
};
#endif
```

```
#include "..\..\CABECERAS\ORDENARNUMEROS\OrdenarNumeros.h"
void OrdenarNumeros::introducirDatos()
{
                elVisualizador.solicitarNumeros(elVectorNumeros);
                std::sort(elVectorNumeros.begin(),elVectorNumeros.end());}
}
void OrdenarNumeros::visualizarResultados()
{
                elVisualizador.visualizarResultados(elVectorNumeros);
}

int main()
{
                OrdenarNumeros elOrdenar;
                elOrdenar.introducirDatos();
                elOrdenar.visualizarResultados();
                return 0;
}
```

Máquina expendedora

Diseñar el programa de una máquina expendedora de bebidas, de manera que recibe el producto seleccionado y las monedas que entran en el cajero desde un autómata programable. La aplicación debe de retornar la lista de monedas mínimas a entregar al usuario.

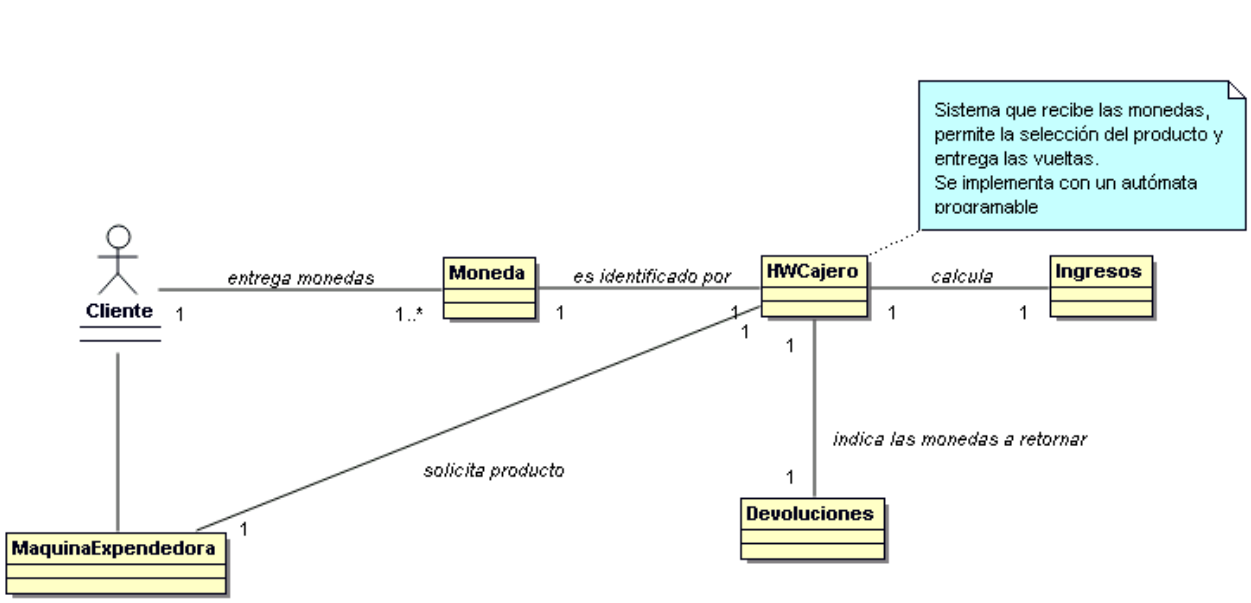
Primero se realizará la jerarquía a dos niveles de las principales características:

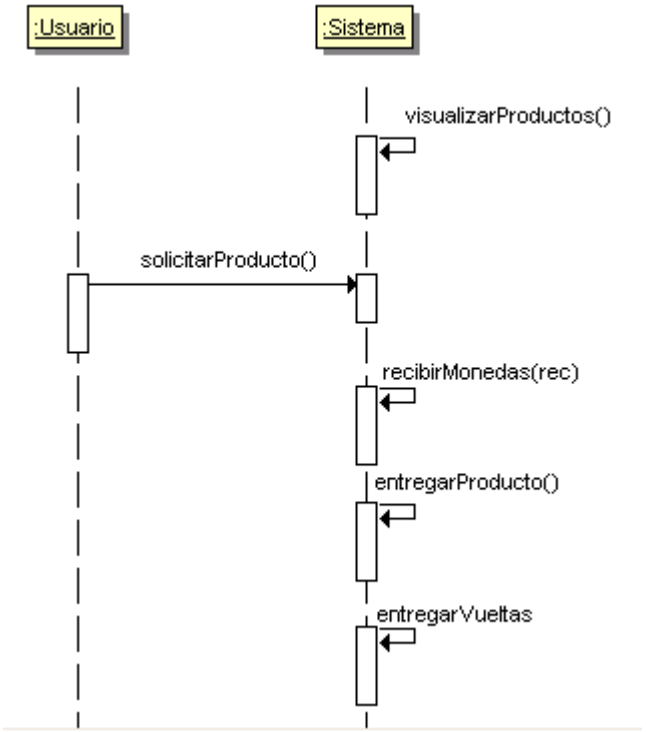
1. El sistema debe de mostrar los productos que ofrece y sus precios
    1.1 El usuario puede elegir uno de los productos ofertados
2. El sistema debe de reconocer las monedas que se le entrega
    2.1 Debe de evaluar el crédito del cliente.
3. El sistema entregará el producto cuando el cliente tenga suficiente crédito.
    3.1 El sistema devolverá las vueltas cuando hubiera exceso de crédito.

Seguidamente se procederá a rellenar los documentos de Visión y Alcance, glosario y el caso de uso EBP: *SolicitarProducto*.

Máquina expendedora

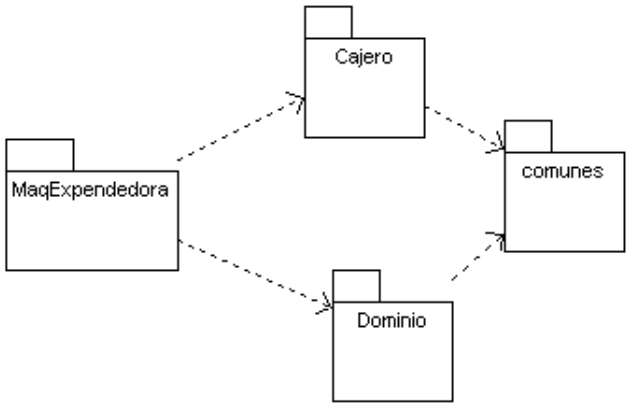
- En el AOO se realizará el modelo del dominio y el DSS:

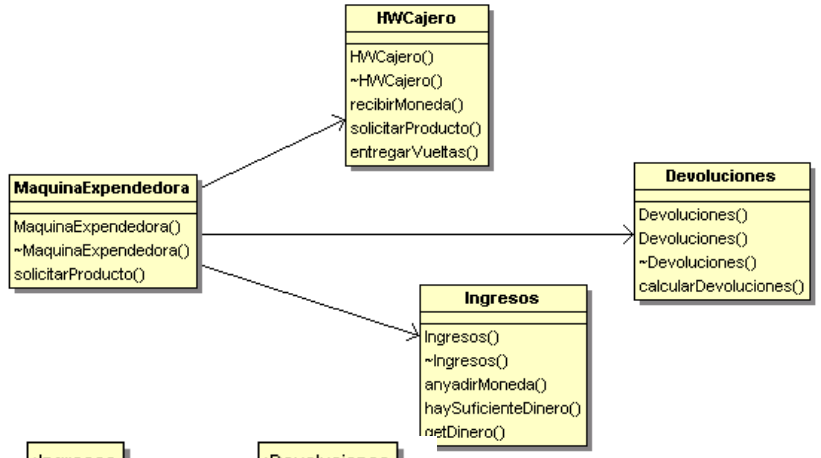


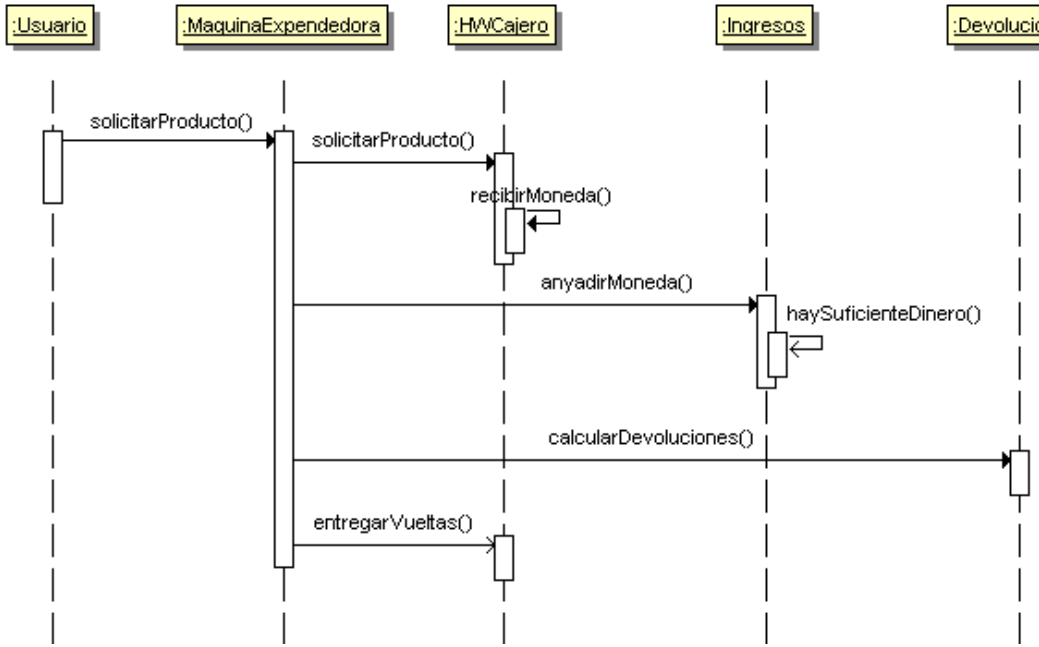


Máquina expendedora

- Durante DOO se planteará la solución lógico a través de la vista de gestión, diagrama de clases de diseño, DCD, y los diagramas de interacción:







Máquina expendedora

- Código de test (implementar las clases)

```
#include "../Cajero/HWCajero.h"
#include "../Dominio/Ingresos.h"
#include "../Dominio/Devoluciones.h"
class MaquinaExpendedora
{
public:
        MaquinaExpendedora();
        virtual ~MaquinaExpendedora();
        void solicitarProducto();
};
////////////////////////////////
void MaquinaExpendedora::solicitarProducto()
{
        HWCajero miCajero;
        Dinero elPrecioProducto = miCajero.solicitarProducto();
        Ingresos miDinero;
        while(miDinero.haySuficienteDinero(elPrecioProducto) == false)
                        miDinero.anyadirMoneda(miCajero.recibirMoneda());
        Devoluciones elDineroEntregar(miDinero.getDinero(),elPrecioProducto);
        std::vector<Moneda> laListaMonedas;
        elDineroEntregar.calcularDevoluciones(laListaMonedas);
        miCajero.entregarVueltas(laListaMonedas);
        miCajero.entregarProducto();
}
////////////////////////////
#include <iostream>
int main()
{
        MaquinaExpendedora laMaquinaExpendedora;
        bool continuar = true;char opcion;
        while(continuar){
                        laMaquinaExpendedora.solicitarProducto();
                        std::cout<<"Nuevo producto (s/n): ";
                        std::cin>>opcion;
                        continuar = (opcion == 'n') || (opcion == 'N') ? false :true;
        }
        return 0;
}
```