3. Indicar la salida por pantalla (2 puntos-15 minutos)

```
#include <iostream.h>
class Tiempo
{
public:
Tiempo ():hora(0), minuto(0){
        cout<<"CONTRUCTOR POR DEFECTO"<<endl;
        };
 Tiempo(int h, int m) : hora(h), minuto(m) {
         cout<<"CONSTRUCTOR NORMAL"<<endl;
        };
 void Mostrar(){cout << hora << ":" << minuto << endl;};
 Tiempo operator+(Tiempo h);
 Tiempo operator+(int mins);
private:
 int hora;
 int minuto;
};

Tiempo Tiempo::operator+(Tiempo h)
{
 Tiempo temp;
 temp.minuto = minuto + h.minuto;
 temp.hora = hora + h.hora;
 if(temp.minuto >= 60)
        {
        temp.minuto -= 60;
        temp.hora++;
        }
 cout<<"SUMA TIEMPO"<<endl;
 return temp;
}
Tiempo Tiempo::operator +(int mins)
{
 Tiempo temp;
 temp.minuto=minuto+mins;
 temp.hora=hora+temp.minuto/60;
 temp.minuto=temp.minuto%60;
 cout<<"SUMA MINUTOS"<<endl;
 return temp;
}

void  main(void)
 {
 Tiempo Ahora(12,24), T1(4,45);
 T1 = Ahora + T1;
 T1.Mostrar();
 (Ahora+45).Mostrar();
 }
```

Impresión por pantalla
CONSTRUCTOR NORMAL
CONSTRUCTOR NORMAL
CONSTRUCTOR POR DEFECTO
SUMA TIEMPO
17:9
CONSTRUCTOR POR DEFECTO
SUMA MINUTOS
13:9

4. Ejercicio de programación (2,5 puntos-50 minutos)
Dada la declaración de la clase base abstracta vehiculo:

```
#include<iostream.h>

class vehiculo
{
  private:
      double coste;
      int anio;
  public:
      vehiculo (double = 0.0, int = 0);
      double obtenerCoste();
      int obtenerAnio ();
      virtual void escribirNombre () = 0;
};

vehiculo::vehiculo(double c, int a):coste(c), anio (a)
{
  //cuerpo vacío
}
double vehiculo::obtenerCoste()
{
  return coste;
}
int vehiculo::obtenerAnio()
{
  return anio;
}
```

Se pide:

a) Defina dos clases derivadas llamadas respectivamente vehiculoTerrestre y vehiculoAereo que deberán concretar la clase vehiculo anterior. Ambas clases derivadas vehiculoTerrestre y vehiculoAereo deberán contener lo siguiente:
   i. Los atributos kilometraje y horas de vuelo respectivamente.
   ii. Los métodos para obtener y establecer esos atributos (típicos métodos set y get de modificación y acceso a los atributos privados/protegidos)
   iii. Los constructores (recibirán tres parámetros cada uno coste, anio y kilometraje/horas de vuelo con valores por defecto).
   iv. La implementación del método `escribirNombre` de manera que este muestre por pantalla el siguiente mensaje: Coste: xx Anio: yy Kilometraje: zz si se trata de un vehículoTerrestre ó Coste: xx Anio: yy Horas de vuelo: zz en el caso de un vehiculoAereo. Los valores xx, yy y zz corresponde al valor de los respectivos atributos del objeto.

b) Escriba una función de prueba *main()* que defina un vector de 2 punteros capaz de almacenar referencias a objetos de las clases derivadas indistintamente. Almacenar en el primer puntero un vehículoTerrestre cuyo coste sea de 25.5, fabricado en el 2006 y con 1000 kilómetros y un vehiculoAereo en el segundo cuyo coste sea de 987.6, fabricado en el 2005 y con 560 horas de vuelo.
c) Recorra el vector mostrando los atributos de cada vehiculo.

Solución:

```
//clase vehiculo terrestre
class vehiculoTerrestre: public vehiculo
{
  private:
       double kilometraje;
  public:
       vehiculoTerrestre(double c = 0.0, int a = 0, double k = 0.0);
       void establecerKilometraje(double);
       double obtenerKilometraje();
       virtual void escribirNombre ();
};
vehiculoTerrestre::vehiculoTerrestre(double c, int a, double k): vehiculo(c,a)
{
       establecerKilometraje(k);
}
vehiculoTerrestre::establecerKilometraje(double k)
{
       Kilometraje = k;
}
double vehiculoTerrestre::ObtenerKilometraje()
{
       return Kilometraje;
}
void vehiculo::escribirNombre()
{
       cout << “Coste:” << obtenerCoste() << “Anio:” << obtenerAnio() <<
       “Kilometraje:” << obtenerKilometraje() << endl;
}

//clase vehiculo aereo
class vehiculoAereo: public vehiculo
{
  private:
       double HorasVuelo;
  public:
       vehiculoAereo(double = 0.0, int = 0, double = 0.0);
       void establecerHorasVuelo(double);
       double obtenerHorasVuelo();
       virtual void escribirNombre () const;
};
vehiculoAereo::vehiculoAereo(double c, int a, double h):vehiculo(c,a)
{
       establecerHorasVuelo(h);
}
vehiculoAereo::establecerHorasVuelo(double h)
{
       HorasVuelo= h;
}
double vehiculoAereo::ObtenerHorasVuelo()
{
       return HorasVuelo;
}
void vehiculo::escribirNombre() const
{
       cout << “Coste:” << obtenerCoste() << “Anio:” << obtenerAnio() << “Horas
       de vuelo:” <<obtenerHorasVuelo() << endl;
}

int main()
{      vehiculo *vectorVehiculo[2];
       vectorVehiculo[0]= new vehiculoTerrestre (25.5, 2006, 1000);
       vectorVehiculo[1]= new vehiculoAereo (987.6, 2005, 560);
       for (int i=0;i<2;i++)
              vectorVehiculo[i] -> escribirNombre();
}
```

5. Problema de Análisis y Diseño Orientado a Objetos (2.5 puntos - 50 minutos)

Se tiene el siguiente código, que sirve para realizar una representación gráfica de una pequeña 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;
};
```

```
//Casa.h

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

class Casa
{
public:
	Casa();
	void Dibuja();
	void SetPos(float x,float
y);

protected:
	Techo techo;
	Bloque planta;
};
```

```
//casa.cpp

#include "Casa.h"

Casa::Casa()
{
	planta.SetBase(1);
	planta.SetAltura(0.5);
	techo.SetBase(1.1);
}
void Casa::Dibuja()
{
	planta.Dibuja();
	techo.Dibuja();
}
void Casa::SetPos(float x, float y)
{
	planta.SetPos(x,y,0);
	techo.SetPos(x,y,planta.GetAltura());
}
```

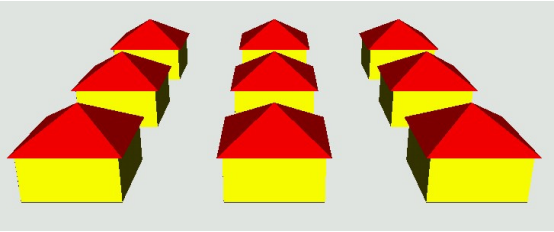
```
//Urbanizacion.h

#include "Casa.h"

class Urbanizacion
{
public:
	Urbanizacion();
	virtual ~Urbanizacion();
	void Dibuja();

protected:
	Casa* casa[100];
	int num_casas;
};
```

```
//Urbanizacion.cpp
#include "Urbanizacion.h"
#include "Casa.h"

Urbanizacion::Urbanizacion()
{
	int i,j,num_casas=0;
	for(i=0;i<3;i++)
	  for(j=0;j<3;j++)
	{
	 casa[num_casas]=new Casa();
	 casa[num_casas]->SetPos(3*i,3*j);
	 num_casas++;
	}
}

Urbanizacion::~Urbanizacion()
{
	for(int i=0;i<num_casas;i++)
		delete casa[i];
}

void Urbanizacion::Dibuja()
{
	for(int i=0;i<num_casas;i++)
		casa[i]->Dibuja();
}
```

El código que instancia las anteriores clases es el siguiente:

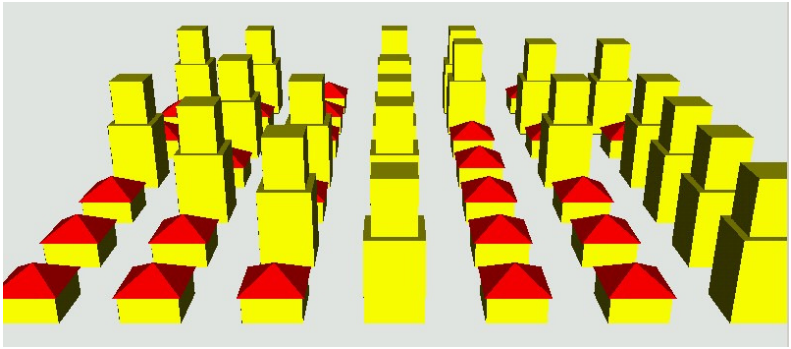
```
#include "Urbanizacion.h"
void main()
{
	Urbanizacion urbe;
	urbe.Dibuja();
}
```

con el siguiente resultado grafico:



Se pide:

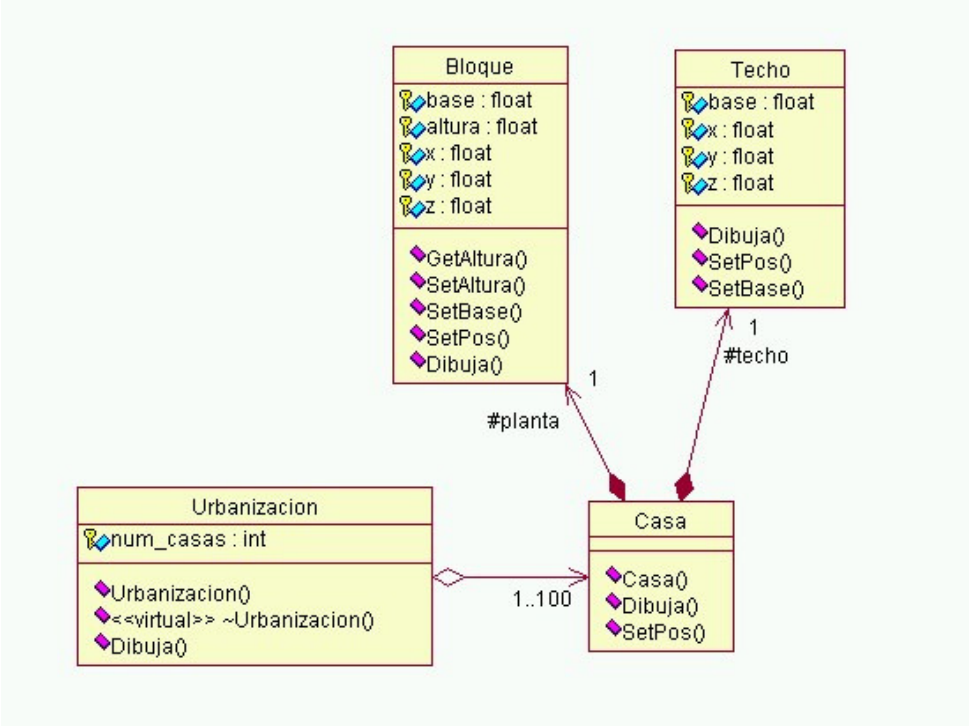
1) Diagrama de Clases de Diseño del código suministrado (ingeniería inversa). **1.5 puntos**
2) Diagrama de secuencias que ilustre la evolución de las llamadas de dibujo, arrancando en la llamada del main *urbe.Dibuja()* Nota: se puede representar el main como un objeto si se desea. **1.5 puntos**
3) Realizar un diseño (Diagrama de Clases de Diseño) con una solución que permita representar en vez de una urbanización una ciudad, en la que puede haber tanto casas como las anteriores como rascacielos. Por simplicidad se puede asumir que cada rascacielos está formado por 2 bloques, uno situado encima del otro, tal como se muestra en la figura siguiente. **4 puntos**

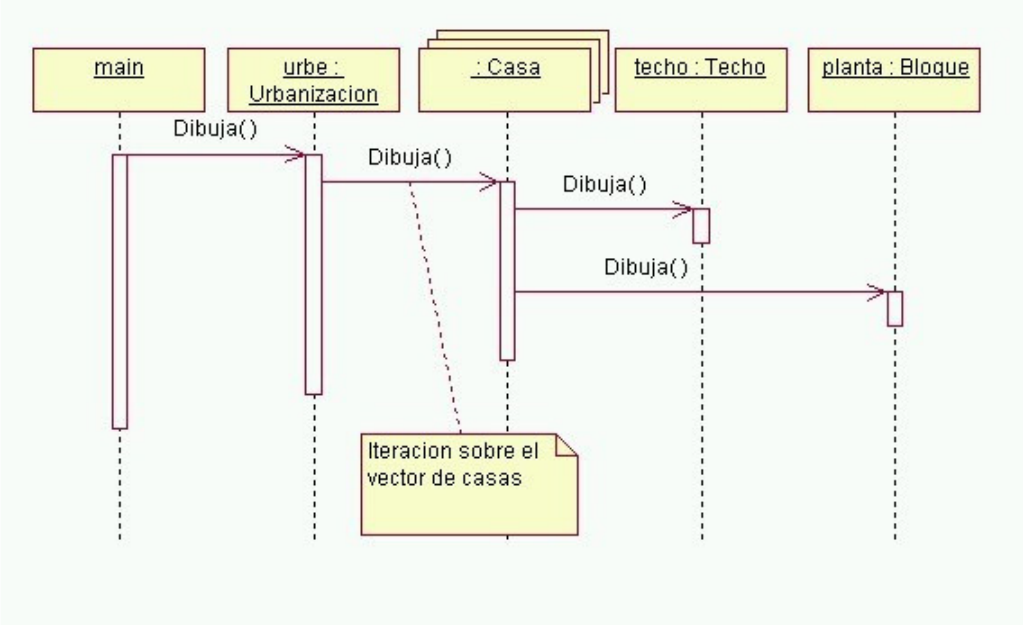


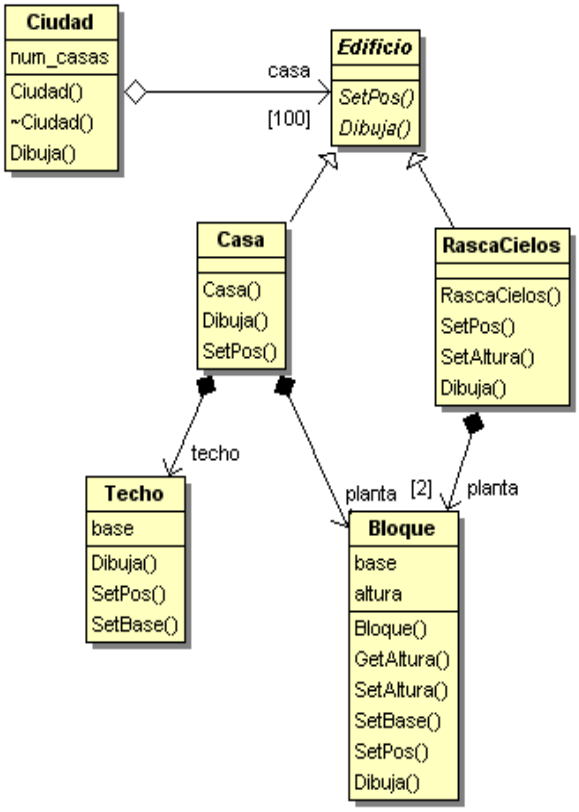
4) Implementación en C++ del diseño del punto anterior (solo es necesario el código nuevo o diferente, siendo necesario definir únicamente los cuerpos de las funciones de dibujo, así como el main) **3 puntos**

Publicación de las notas: 22-09-2010
Revisión: 23-09-2010

SOLUCION:


Bloque
base : float
altura : float
x : float
y : float
z : float
GetAltura()
SetAltura()
SetBase()
SetPos()
Dibuja()
Techo
base : float
x : float
y : float
z : float
Dibuja()
SetPos()
SetBase()
1
#techo
1
#planta
Urbanizacion
num_casas : int
Urbanizacion()
<<virtual>> ~Urbanizacion()
Dibuja()
1..100
Casa
Casa()
Dibuja()
SetPos()


main
urbe : Urbanizacion
: Casa
techo : Techo
planta : Bloque
Dibuja()
Dibuja()
Dibuja()
Dibuja()
Iteracion sobre el vector de casas




```
class Edificio
{
public:
      virtual void SetPos(float x,float y)=0;
      virtual void Dibuja()=0;
};
```

```
class RascaCielos  :public Edificio
{
public:
      RascaCielos();
      void SetPos(float x,float y);
      void SetAltura(float alt);
      void Dibuja();

protected:
      Bloque planta[2];
};
```

```
class Casa  :public Edificio
{
public:
      Casa();
      void Dibuja();
      void SetPos(float x,float y);

protected:
      Techo techo;
      Bloque planta;
};
```

```
#include "Edificio.h"   // Added by ClassView

class Ciudad
{
public:
      Ciudad();
      virtual ~Ciudad();
```

```
	void Dibuja();

protected:
	Edificio* casa[100];
	int num_casas;
};
```

```
void RascaCielos::Dibuja()
{
	planta[0].Dibuja();
	planta[1].Dibuja();
}
```

```
void Ciudad::Dibuja()
{
	for(int i=0;i<num_casas;i++)
		casa[i]->Dibuja();
}
```