3. Indicar la salida por pantalla (2 puntos–20 minutos)

```
#include <iostream.h>
class EUITIelemento;
class EUITIpila
{
int num;
   EUITIelemento *pila[10];
public:
   EUITIpila():num(0){};
   bool push(EUITIelemento *in){
        if(num>=10)return false;
        pila[num++]=in;
        return true;
        }
   EUITIelemento *pop(){
        cout<<"pop!"<<endl;
        if(num==0)return 0;
        return pila[--num];
        }
   int getNum(){return num;}
};
class EUITIelemento
{
protected:
   float valor;
public:
   virtual bool ejecuta(EUITIpila *)=0;
   float getValor(EUITIpila *p=0){
        ejecuta(p);
        return valor;
        }
};
class EUITIvalor:public EUITIelemento
{
public:
   bool ejecuta(EUITIpila *p){return true;}
   EUITIvalor(float v){valor=v;}
};
class EUITIsuma:public EUITIelemento
{
public:
  bool ejecuta(EUITIpila *p){
     EUITIelemento *aux1=p->pop();
     if(aux1==0)return false;
     float v1=aux1->getValor(p);
     EUITIelemento *aux2=p->pop();
     if(aux2==0)return false;
     float v2=aux2->getValor(p);
     valor=v1+v2;
     cout<<"suma de "
    <<v1<<"+"<<v2<<"="<<(v1+v2)<<endl;
     delete aux1;
     delete aux2;
     return true;
     }
};

void main()
{
EUITIpila calculadora;
calculadora.push(new EUITIvalor(5));
calculadora.push(new EUITIvalor(4));
calculadora.push(new EUITIvalor(3));
calculadora.push(new EUITIsuma);
calculadora.push(new EUITIsuma);
calculadora.push(new EUITIvalor(2));
calculadora.push(new EUITIsuma);
calculadora.pop()->getValor(&calculadora);
}
```

línea	Impresión por pantalla
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

4. Ejercicio de programación (2,5 puntos-50 minutos)
Dadas las siguientes clases en C++:

```
class Instrumento
{
protected:
int peso_;
char* nombre_;
public:
Instrumento(): peso_(0)
{
nombre_ = new char[8];
strcpy(nombre_,"Ninguno");
}
~Instrumento()
{
delete nombre_;
}
int peso() const
{
return peso_;
}

void peso(const int& peso)
{
peso_ = peso;
}
char* nombre() const
{
return nombre_;
}
void nombre(const char* nombre)
{
delete nombre_;
if (0 == nombre)
{
nombre_ = 0;
}
else
{
int len = strlen(nombre);
nombre_ = new char[len + 1];
strcpy(nombre_,nombre);
}
}
void mostrar ()
{
cout << "Mostrando Clase Instrumento con nombre:" << nombre_ << endl;
}
virtual void tocar_nota()=0;
};

class Guitarra : public Instrumento
{
private:
int cuerdas_;
public:
Guitarra() : Instrumento(), cuerdas_(6) {};
~Guitarra() {};
void mostrar()
{
cout << "Mostrando Clase Guitarra con nombre:" << nombre_ << endl;
};
void tocar_nota()
{
cout << "Tocando nota con Guitarra:" << endl;
};
};
```

Hacer un programa principal que realice lo siguiente:

a) Crear un array de 6 punteros a Instrumento.
b) Inicializar los punteros del array con objetos de la clase Guitarra.
c) Aplicar a los seis elementos la función tocar_nota() ¿Qué sale por pantalla?
d) Aplicar a los seis elementos la función mostrar() ¿Qué sale por pantalla?
e) ¿Podríamos crear un objeto de la clase Instrumento directamente? Razonar la respuesta.
f) ¿Sería necesario crear un constructor copia para la clase Instrumento? Razonar la respuesta.

5. Problema de Análisis y Diseño Orientado a Objetos (2.5 puntos – 50 minutos)
Para el código adjuntado se pide:

a. Ingeniería inversa: Diagrama de clases.
b. Ingeniería inversa: Diagrama de secuencia.
c. Resultado de su ejecución en la consola.
d. Indicar los patrones GRASP empleados en este patrón.
e. Diseñar e implementar el servicio ***rotar( ),*** tal que

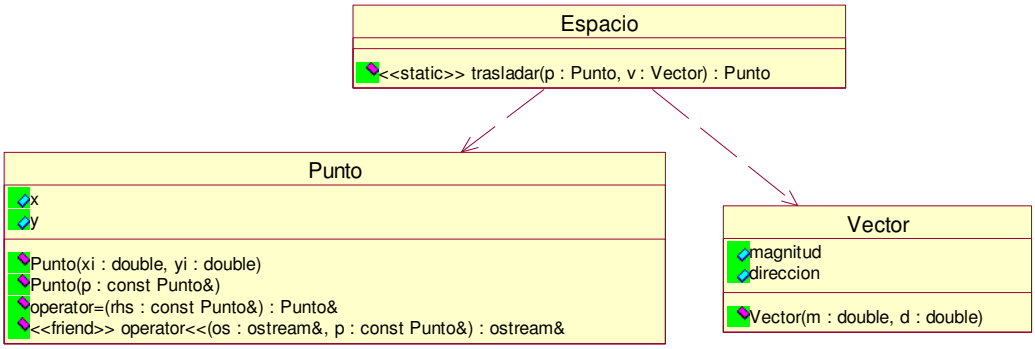
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

Empléese sobre el punto *p2*.

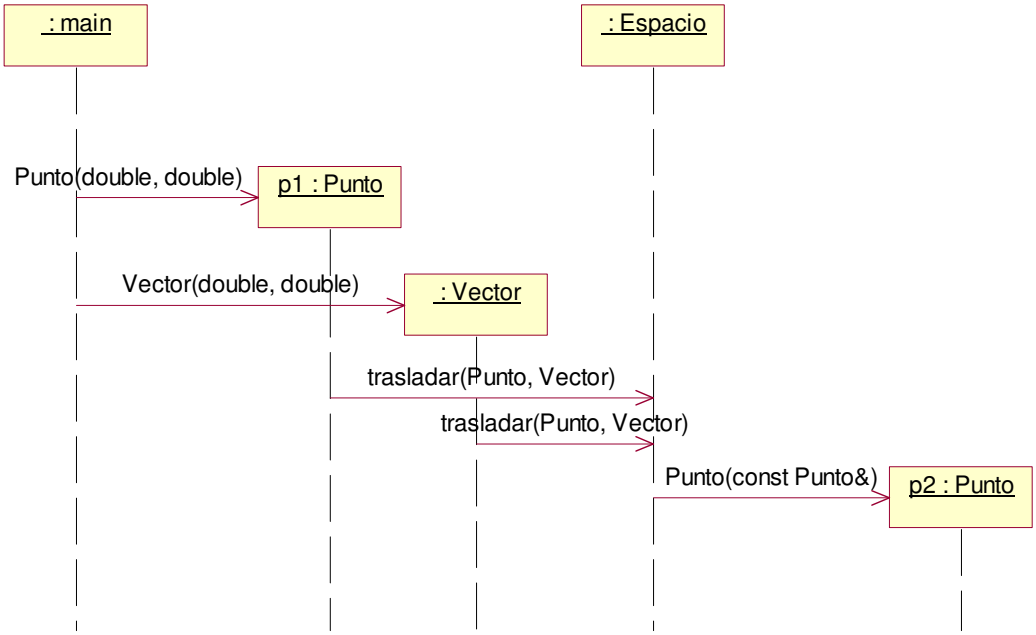
```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

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;
}
```

Resolución

a)



b)



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

d) 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.

e)

```
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;
}
```