UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Diciembre 2015**

Calificación

2. Problema de Análisis y Diseño Orientado a Objetos (10 puntos - 30 minutos)

En una primera versión de un juego de ajedrez se ha implementado dos piezas y un tablero de madera, se pide.

1. Diagrama Clase de Diseño e indique los patrones empleados (5 puntos).
2. Implementación de las clases *Pieza, Rey, Peon, Tablero, TableroMadera, Factoria* (5 puntos).

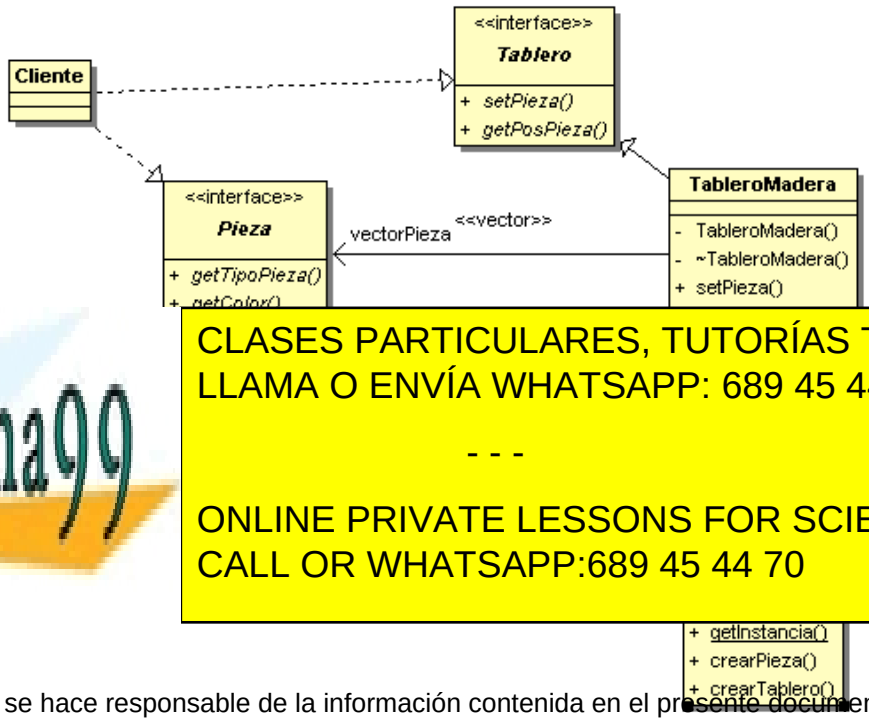
```
#include <vector>
#include <iostream>
using namespace std;

typedef enum{REY,PEON} TipoPieza;
typedef enum{MADERA} TipoTablero;
typedef enum{BLANCO,NEGRO} Color;

class Pieza{…};
class Rey{…};
class Peon {…};
class Tablero{…};
class TableroMadera{…};
class Factoria{…};

int main()
{
 Factoria &factoria= Factoria::getInstancia();
 Pieza *p1 = factoria.crearPieza(REY,BLANCO);
 Pieza *p2 = factoria.crearPieza(PEON,NEGRO);
 Tablero *tablero = factoria.crearTablero(MADERA);
 tablero->setPieza(p1,'e',8);
 tablero->setPieza(p2,'a',2);
 cout << "Pieza: " << p1->getTipoPieza() << " " <<
  p1->getColor() << " " << tablero->getPosPieza(p1) << endl;
 cout << "Pieza: " << p2->getTipoPieza() << " " <<
  p2->getColor() << " " << tablero->getPosPieza(p2) << endl;
 delete tablero;
}

//////////////////////////////////////////////////////////////
Ejecución del programa
//////////////////////////////////////////////////////////////
Pieza: Rey Blanco e8
Pieza: Peon Negro a2
```



```
class Pieza{
public:
 virtual char * getTipoPieza()=0;
 virtual char * getColor()=0;
};


class Rey : public Pieza{
 Color color;
 friend class Factoria;
 Rey(Color c):color(c){}
public:
 char * getTipoPieza() {return "Rey";}
 char * getColor() {
      if(color==BLANCO) return "Blanco";
      else return "Negro";
      }
};

class Peon : public Pieza{
 Color color;
 friend class Factoria;
 Peon(Color c):color(c){}
public:
 char * getTipoPieza() {return "Peon";}
 char * getColor() {
      if(color==BLANCO) return "Blanco";
      else return "Negro";
      }
};


class Tablero{
public:
 virtual void setPieza(Pieza *,char,int)=0;
 virtual char * getPosPieza(Pieza *)=0;
};

class PosPieza {
 char colum;
 int fila;
 char res[3];
public:
 PosPieza(int f,char c):fila(f),colum(c){}
 char *getPos() {
      res[0]=colum;
      res[1]=(char)(fila+48);
      res[2]='\0';
      return res;
 }
};
```

```
class TableroMadera : public Tablero{
 vector<Pieza *>vectorPieza;
 vector<PosPieza *> vectorPiezaPos;
 friend class Factoria;
 TableroMadera(){}
 ~TableroMadera(){
  for(int i=0; vectorPieza.size(); i++){
   if(vectorPieza[i]) delete vectorPieza[i];
   if(vectorPiezaPos[i])deletevectorPiezaPos[i];
      }
}
public:
 void setPieza(Pieza *p,char c,int f){
  vectorPieza.push_back(p);
  vectorPiezaPos.push_back(new PosPieza(f,c));
 }
 char * getPosPieza(Pieza *p){
  int i=0;
  while(p != vectorPieza[i]) i++;
  return vectorPiezaPos[i]->getPos();
 }
};


class Factoria
{
 Factoria(){}
public:
 static Factoria& getInstancia(){
   static Factoria unicaInstancia;
   return unicaInstancia;
  }
Pieza* crearPieza (TipoPieza tipo, Color c){
  if(tipo == REY) return new Rey(c);
  else if(tipo == PEON ) return new Peon(c);
  else return NULL;
 }
 Tablero* crearTablero (TipoTablero tipo){
   if(tipo == MADERA) return new TableroMadera;
   else return NULL;
   }
};
```

escuela técnica superior de ingeniería y diseño industrial

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Diciembre 2015

Calificación

3. Problema de Sistemas Operativos (15 puntos - 30 minutos)

Se desea ejecutar sobre un sistema UNIX el fragmento de código adjunto.
En el código del programa, se proyecta el fichero '/tmp/puntos en memoria como una zona de datos compartida. Se pide:

1. Escriba el código para realizar la sincronización necesaria utilizando el mecanismo de **tuberías**. El código de sincronización debe hacer que ejecute primero el proceso padre, después el proceso hijo y así sucesivamente. Indique en qué líneas va cada una de las sentencias. (10 puntos)

   Para poder asegurar alternancia entre padre e hijo es necesario al menos dos tuberías, a través de las cuales se van escribiendo y leyendo un carácter que hace de testigo para acceder a la zona de memoria compartida.

2. Indique qué valores se escriben por pantalla. Justifique razonadamente la respuesta. (3 puntos)

   x:0, y:0
   x:0, y:0
   x:1, y:2
   x:3, y:5
   x:2, y:4
   x:6, y:10
   x:3, y:6

3. Explique y razone qué otros mecanismos de sincronización se podrían haber utilizado para el mismo propósito. (2 puntos)

```
1  #define MAX_IT 6
2
3  struct punto {
4   int x;
5   int y;
6  };
7
8  struct punto *pp;
9  struct punto *ph;
10
11 int main(void)
12 {
13  int fd;
14  struct stat bstat;
15  int i;
16  void *p;
17  pid_t pid;
    int tuberia[2], tuberia2[2];
    char testigo;

    pipe(tuberia);
    pipe(tuberia2);

    write(tuberia[1], &testigo, 1);
18  fd = open ("/tmp/puntos", O_RDWR); //Fichero existe
19  fstat(fd, &bstat);
20  p= mmap(NULL,bstat.st_size,PROT_READ |
21            PROT_WRITE, MAP_SHARED, fd, 0);
22  close(fd);
23
24  if (fork() != 0){
25    pp = p;
      close(tuberia[1]); close(tuberia2[0]);
26    for ( i=0; i<=MAX_IT; i++){
27     read(tuberia[0], &testigo, 1);
28     pp->x = i;
29     pp->y = i*2;
30     pp++;
31     pp->x = i*3;
42     pp++; // Avanza hasta la siguiente estructura
43     write(tuberia[1], &testigo, 1);
44    }
45    close(tuberia2[0]); close(tuberia[1]);
46 }
```