Ejercicios propuestos de **Programación orientada a objetos con C++**

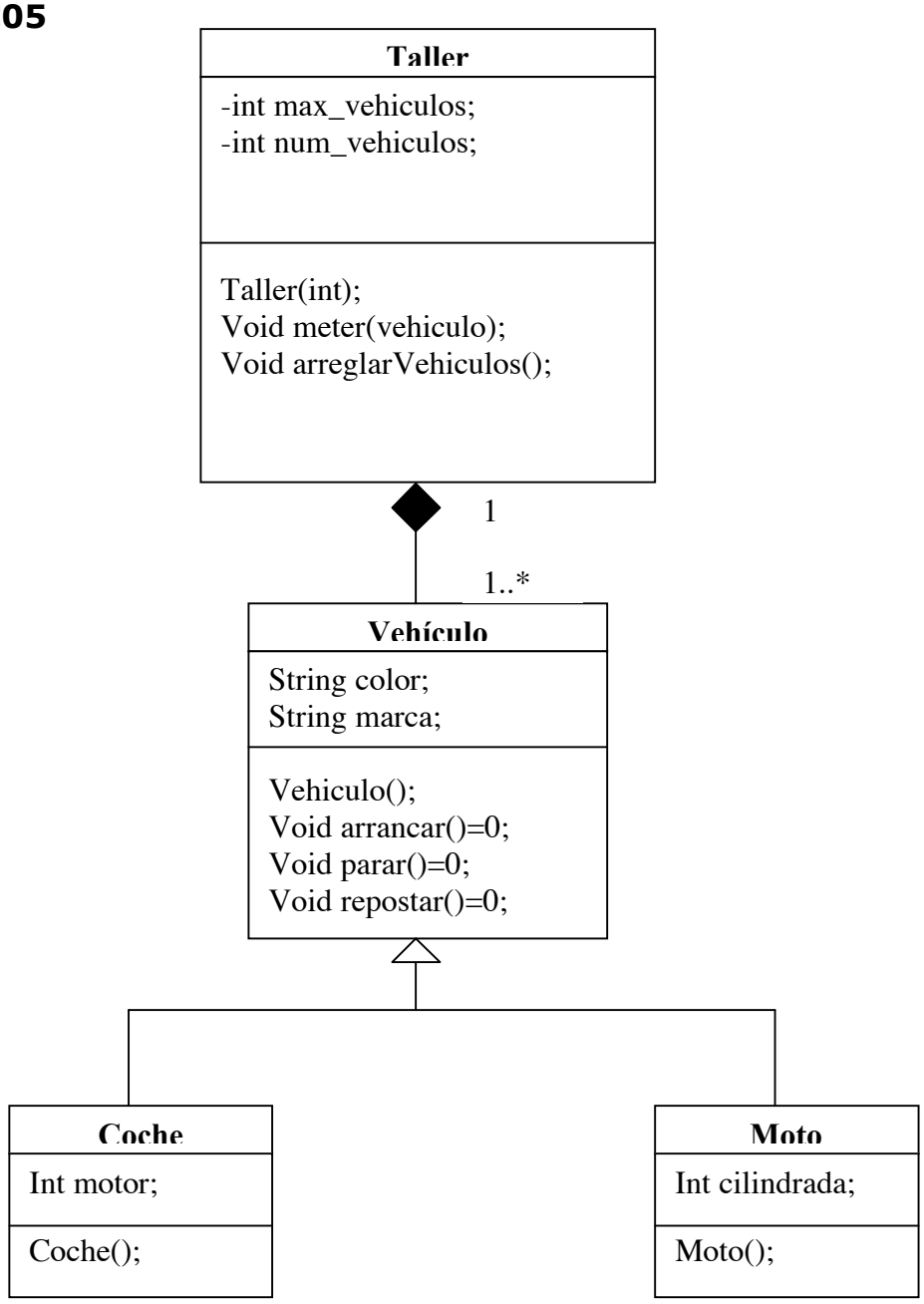
Cristina Cachero
Pedro J. Ponce de León

Departamento de Lenguajes y Sistemas Informáticos

Universidad de Alicante

En cada ejercicio se indica la convocatoria de examen en la que apareció (por ejemplo, DIC05: Convocatoria extraordinaria de Diciembre de 2005).

1. DIC05



(2 puntos) Implementad tanto el .h como el .cpp de la clase taller La función meter (coche o moto) implica introducir el coche en el taller y la función arreglarCoches o arreglarMotos implica recorrer todos los coches y motos que haya y arrancarlos.

2. SEP05
(1,5 puntos) Declarad (sólo cabecera), para cada una de las siguientes sentencias A, B y C, qué haría falta añadir a una clase Matriz (implementada mediante undoble puntero a enteros) para que funcionasen correctamente.

```
int main(){
     Matriz c;
     /*A*/ c[3,5]; //devuelve el elto. En la pos. 3,5
     /*B*/ c[3][5]; // idem
     /*C*/ c(3,5); // idem
     return (0);
};
```

3. SEP05 (2,5 puntos) Dadas las siguientes definiciones de clase

```
//---------------------------------------------------------------------
// CLASE BASE
//---------------------------------------------------------------------
class cBase {
  private:
    int objeto_base_1;
    int objeto_base_2;
  public:
    cBase();
    virtual ~cBase();

    void funcion_A();
    void funcion_B();
    virtual void funcion_C();
    virtual void funcion_D() = 0;
};

cBase :: cBase() : objeto_base_1(1 ),
                   objeto_base_2(2 )
{ cout << "Constructor de BASE" << endl; }

cBase :: ~cBase()
{  cout << "Destructor de BASE" << endl;}

void cBase :: funcion_A()
{  cout << "Funcion A de BASE" << endl;}

void cBase :: funcion_B()
{  cout << "Funcion B de BASE" << endl;}

void cBase :: funcion_C()
{  cout << "Funcion C de BASE" << endl;}



//---------------------------------------------------------------------
// CLASE DERIVADA
//---------------------------------------------------------------------
class cDerivada : public cBase {
  private:
```

```
    int objeto_deriv_1;
    int objeto_deriv_2;
  public:
    cDerivada();
    ~cDerivada();

    void funcion_A();
    void funcion_D();
};

cDerivada :: cDerivada() : cBase(),
                           objeto_deriv_1(3),
                           objeto_deriv_2(4)
{ cout << "Constructor de DERIVADA" << endl; }

cDerivada :: ~cDerivada()
{  cout << "Destructor de DERIVADA" << endl;}

void cDerivada :: funcion_A()
{  cout << "Funcion A de DERIVADA" << endl;}

void cDerivada :: funcion_D()
{  cout << "Funcion D de DERIVADA" << endl;}
```

se pide:

a. Especificad qué métodos invoca el siguiente programa.

```
int main( void )
{
  cDerivada derivada;
  cBase *base_ptr = &derivada;

  // funcion de base, redefinida en derivada
  derivada.funcion_A();
  base_ptr->funcion_A();

  // funcion de base, no redefinida en derivada
  derivada.funcion_B();
  base_ptr->funcion_B();

  // funcion virtual de base, no redefinida en derivada
  derivada.funcion_C();
  base_ptr->funcion_C();

  // funcion virtual pura de base, redefinida en derivada
  derivada.funcion_D();
  base_ptr->funcion_D();}
```

4. SEP04

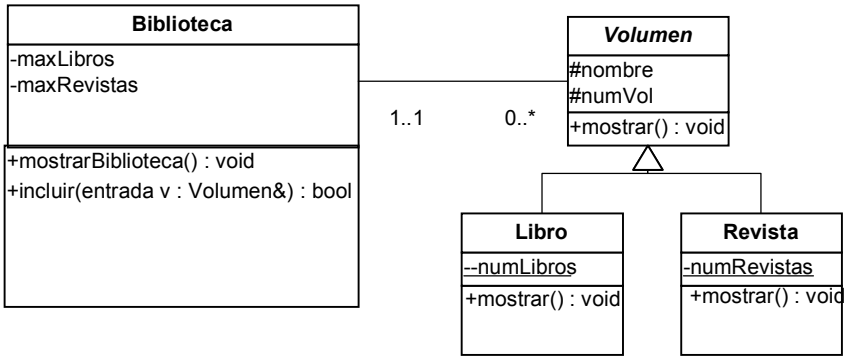


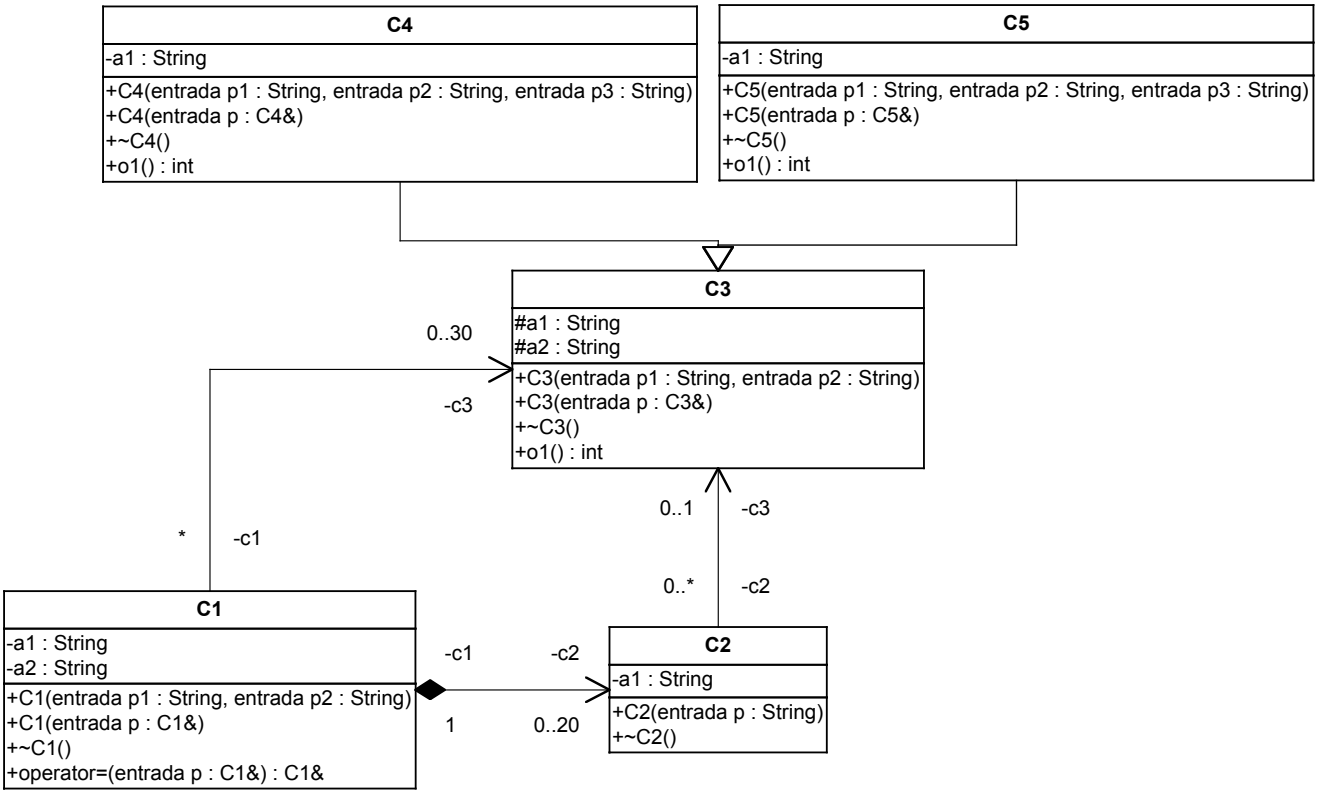
Fig. 2. Diagrama de clases Sistema de Gestión de Bibliotecas

a. Define la función *mostrar()* de la clase **Volumen**, así como la de la clase **Revista** (signatura y cuerpo de la función), teniendo en cuenta que dicha función debe comportarse de manera polimórfica. ¿Qué otra circunstancia tiene que darse para que el tipo de la variable que invoca al método no determine el método a utilizar? (2 puntos)

b. Implementa (signatura y código) una función *incluir(...)* en la clase Biblioteca que permita añadir con el mismo código tanto una revista como un libro que se le pase como parámetro. (1,5 puntos)

5. DIC03

Supongamos que los diseñadores de nuestra empresa nos entregan el diagrama de clases de la figura:.



Se pide:

a) Implementa la declaración (el .h) y el destructor de C1 (2 puntos)

b) Implementa la declaración y el constructor de copia de la clase C4 (1,5 puntos)

---

6. DIC04

1.- Supongamos que tenemos definida la jerarquía de herencia de la Fig.1.:

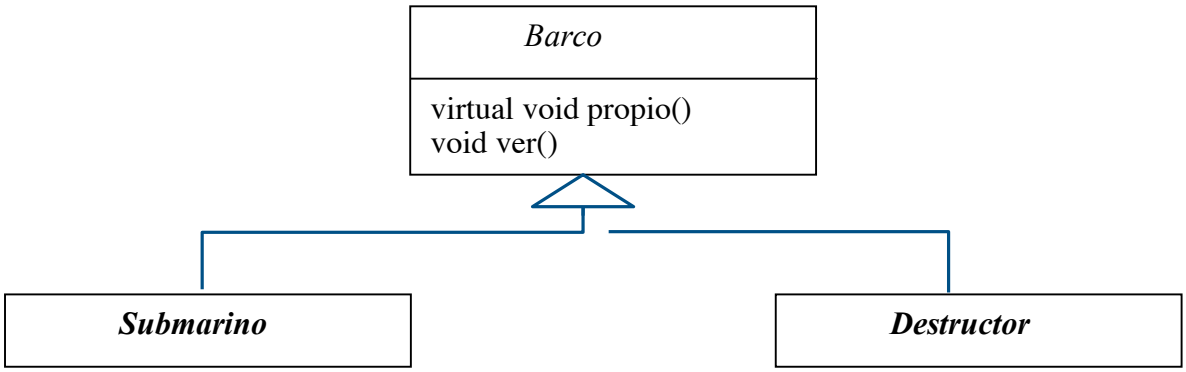


Fig.1: Jerarquía de herencia de Barco

1.a. (2.5 puntos) Especifica qué pasará (a qué clase se referirá la invocación) en los siguientes casos, suponiendo que el método *propio()* está sobreescrito en las clases derivadas.

```
a) Barco *b=new Barco();
   b->propio();
   b->ver();

b) Barco *b=new Submarino();
   b->propio();
   b->ver();

c) Barco *b[]={new Barco(),new Submarino()};
   b[0]->propio();
   b[1]->propio();
   b[0]->ver();

d) Barco *b[]={new Destructor(),new Submarino()};
   b[0]->propio();
   b[1]->propio();
```

e) Supongamos que añadimos el método *ver()* heredado del padre tanto en submarino como en destructor.

```
Barco *b[]={new Destructor(),new Submarino()};
((Destructor *)b[0])->ver();
((Submarino *)b[1])->ver();
b[0]->ver();
b[1]->ver();
```

1.b. (0,5 puntos) Supongamos ahora que la signatura del método propio es la siguiente:

```
virtual void Barco::propio()=0;
```

y que además dicho método se sobrescribe en las clases derivadas. ¿Qué pasaría con el siguiente código?

```
Barco *b[]={new Barco(),new Submarino()};
b[0]->propio();
b[0]->ver();
b[1]->propio();
b[1]->ver();
```

---

7. FEB07 (4 puntos) A partir del código en C++ que aparece a continuación, contesta a las siguientes preguntas. No olvides hacer referencia en tus respuestas al número de línea involucrado.

a) ¿Existe algún problema que dé como resultado un error de compilación? Justifica brevemente por qué se produce el error o errores e indica qué líneas de código los producen.

b) ¿Qué tipo de polimorfismo se está utilizando en las definiciones de las clases derivadas respecto a sus clases base? Indica, para cada método definido en las clases derivadas, si se trata de redefinición, shadowing o sobrescritura.

c) En el programa principal, indica mediante su signatura completa qué método es ejecutado en cada instrucción antes del comentario TEST.

d) Por último, indica qué salida produce este programa.

```
1: #include <iostream>
2: using namespace std;
3:
4: class Padre{
5:   friend ostream& operator<<(ostream& o, const Padre& p) {
6:     o << "PADRE" << endl; return o;
7:   }
8:   public:
9:     virtual int ejemplo(int a) {}
10:            void ejemplo(int a, int b) {}
11:    virtual int otro(int x) {}
12:            void otro(int a, int b, int c) {}
13:    virtual void print() {}
14:};
15:
16: class Hija : public Padre{
17:   friend ostream& operator<<(ostream& o, const Hija& p) {
18:     o << "HIJA" << endl; return o;
19:   }
20:   public:
21:     int ejemplo (int a) {}
22:     void ejemplo (int a, int b) {}
23:     int otro(int x) {}
24:     float otro(int x,int y) {}
25:     void print() { cout << *this; }
26: };
27:
28: class Nieta: public Hija {
29:   public:
30:     int ejemplo (int a) {}
31: };
32:
33: Hija* Test (Padre *p) {
34:   if (p) p->print(); return dynamic_cast<Hija*>(p);
35: };
36:
37: Padre* Test (Hija *h) {
38:   if (h) h->print(); return h;
39: };
40:
41: int main(){
42:   Nieta n; Padre* p = &n;
43:   p->ejemplo(1);
44:   p->ejemplo(1,2);
45:   p->otro(1);
46:   p->otro(1,2);
47:   p->otro(1,2,3);
48:
49:   Hija *ph = &n;
50:   ph->ejemplo(1);
51:   ph->ejemplo(1,2);
52:   ph->otro(1);
53:   ph->otro(1,2);
54:   ph->otro(1,2,3);
55:
56:   /* TEST */
57:   p=Test(new Padre());
58:   ph=(Hija*)Test(new Hija());
59:   if (p) cout << *p;
60:   if (ph) cout << *ph;
61: }
```

8. FEB07 (3 puntos) Estudia el diagrama UML que se muestra al final de esta pregunta, correspodiente a una implementación parcial de un juego de ajedrez. Implementa el **constructor de copia** de la clase PartidaAjedrez teniendo en cuenta lo siguiente:

- que, al salir de dicho constructor de copia, el nuevo objeto debe replicar el estado de la partida original (posición de las piezas en el tablero y movimientos). Para ello, puedes apoyarte en cualquiera de los demás métodos incluidos en el diagrama de clases.

- que el atributo estático Movimiento::numMovs ha sido redefinido como un atributo de instancia de la clase PartidaAjedrez.

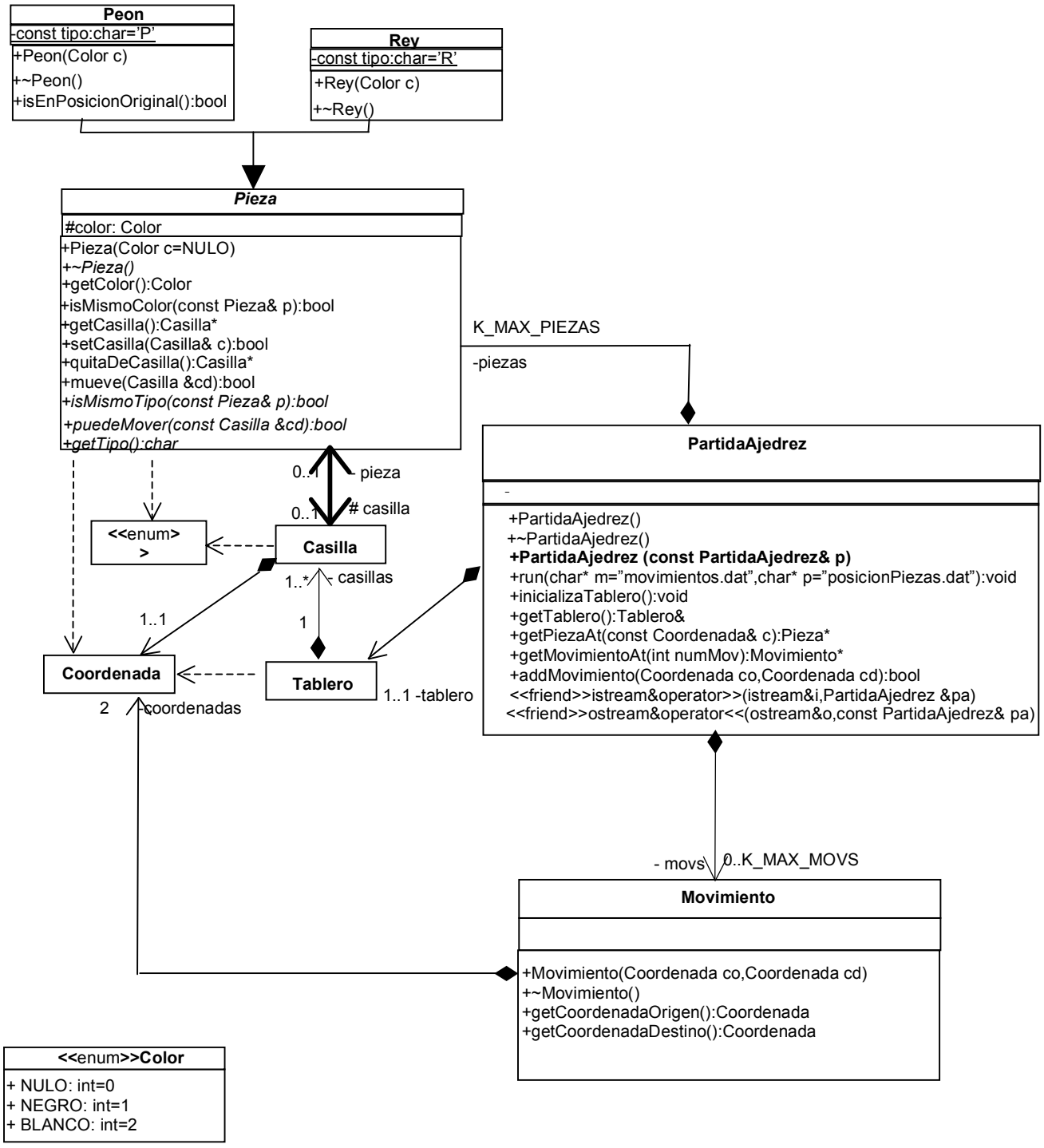
- que las demás clases han sido convenientemente modificadas para tener en cuenta estos cambios en el diseño y que todas ellas implementan explícitamente su forma canónica.

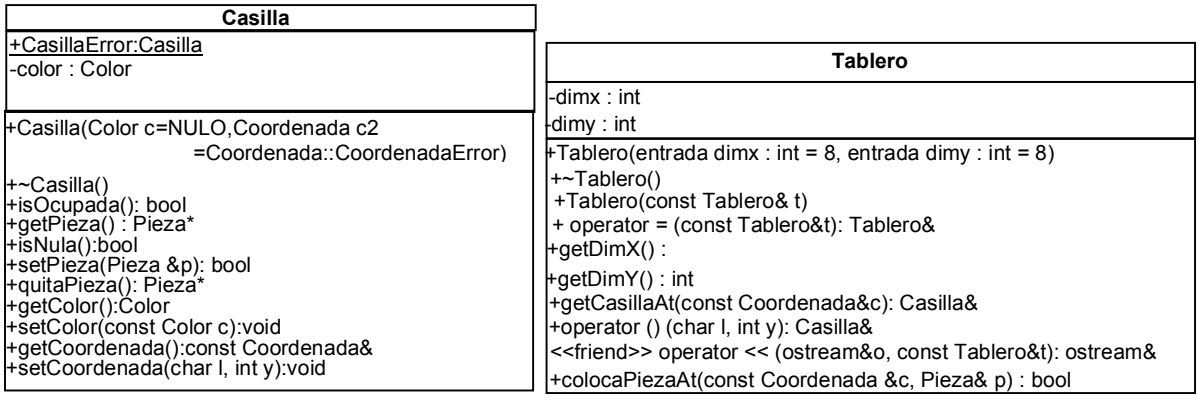
- que el constructor de PartidaAjedrez tiene la siguiente implementación :

```
PartidaAjedrez::PartidaAjedrez():numMovs(0){
  piezas=new Pieza*[K_MAX_PIEZAS];
/*se asume que la pos de las piezas en el array PartidaAjedrez::piezas
no cambia durante la partida*/
  piezas[0] = new Rey(BLANCO);
  piezas[1] = new Rey(NEGRO);

  for (int i=2;i<10;i++)
    piezas[i]=new Peon(BLANCO);

  for (int i=10; i<K_MAX_PIEZAS; i++)
    piezas[i]=new Peon(NEGRO);

  movs=new Movimiento*[K_MAX_MOVS];
  for (int i=0; i<K_MAX_MOVS; i++)
    movs[i]=NULL;
}
```

Ampliación 1 : Reescribe el constructor de copia suponiendo que no se conoce qué tipo de pieza hay en cada posición del array PartidaAjedrez::piezas

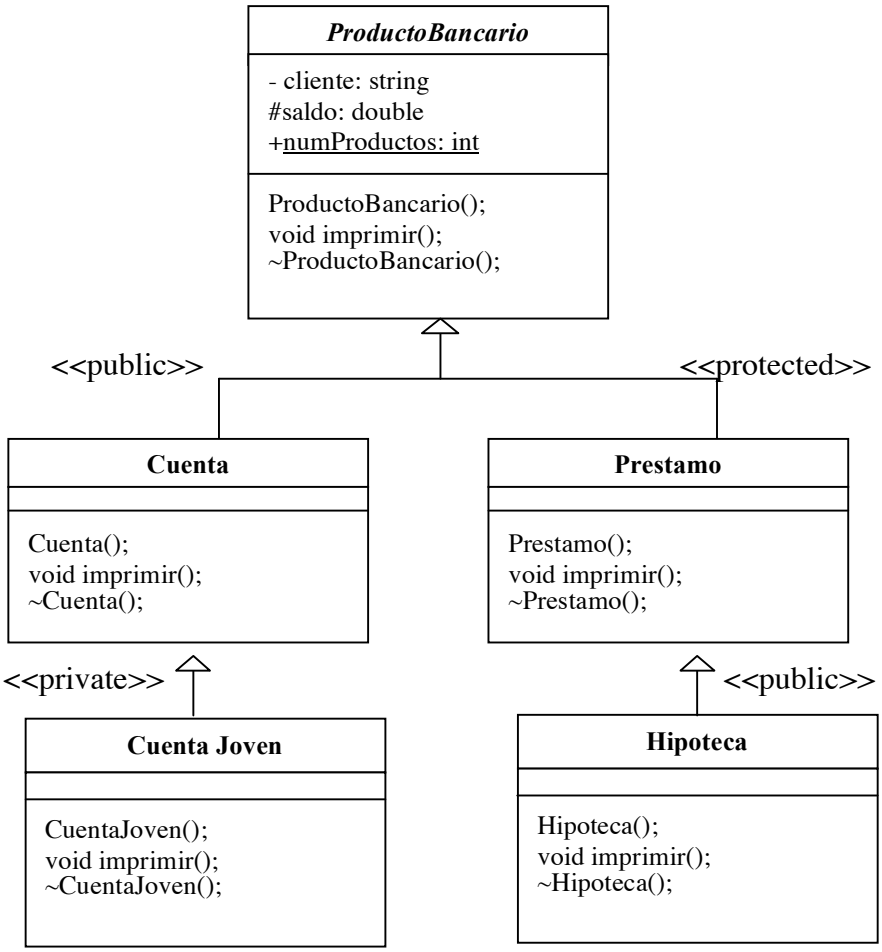
Ampliación 2 : Supón ahora que ni siquiera conoces el tipo de piezas que pueden estar almacenadas en dicho array. ¿Cómo podrías implementar el constructor de copia de PartidaAjedrez, de manera que fuera independiente de las clases de pieza derivadas de Pieza?


Peon
-const tipo:char='P'
+Peon(Color c)
+~Peon()
+isEnPosicionOriginal():bool
Rey
-const tipo:char='R'
+Rey(Color c)
+~Rey()
Pieza
#color: Color
+Pieza(Color c=NULO)
+~Pieza()
+getColor():Color
+isMismoColor(const Pieza& p):bool
+getCasilla():Casilla*
+setCasilla(Casilla& c):bool
+quitaDeCasilla():Casilla*
+mueve(Casilla &cd):bool
+isMismoTipo(const Pieza& p):bool
+puedeMover(const Casilla &cd):bool
+getTipo():char
K_MAX_PIEZAS
-piezas
PartidaAjedrez
+PartidaAjedrez()
+~PartidaAjedrez()
+PartidaAjedrez (const PartidaAjedrez& p)
+run(char* m="movimientos.dat",char* p="posicionPiezas.dat"):void
+inicializaTablero():void
+getTablero():Tablero&
+getPiezaAt(const Coordenada& c):Pieza*
+getMovimientoAt(int numMov):Movimiento*
+addMovimiento(Coordenada co,Coordenada cd):bool
<<friend>>istream&operator>>(istream&i,PartidaAjedrez &pa)
<<friend>>ostream&operator<<(ostream&o,const PartidaAjedrez& pa)
0..1
- pieza
0..1
casilla
<<enum>>
Casilla
1..*
casillas
1..1
1
Coordenada
Tablero
1..1 -tablero
2
-coordenadas
- movs
0..K_MAX_MOVS
Movimiento
+Movimiento(Coordenada co,Coordenada cd)
+~Movimiento()
+getCoordenadaOrigen():Coordenada
+getCoordenadaDestino():Coordenada
<<enum>>Color
+ NULO: int=0
+ NEGRO: int=1
+ BLANCO: int=2


Casilla
+CasillaError:Casilla
-color : Color
+Casilla(Color c=NULO,Coordenada c2
=Coordenada::CoordenadaError)
+~Casilla()
+isOcupada(): bool
+getPieza() : Pieza*
+isNula():bool
+setPieza(Pieza &p): bool
+quitaPieza(): Pieza*
+getColor():Color
+setColor(const Color c):void
+getCoordenada():const Coordenada&
+setCoordenada(char l, int y):void
Tablero
-dimx : int
-dimy : int
+Tablero(entrada dimx : int = 8, entrada dimy : int = 8)
+~Tablero()
+Tablero(const Tablero& t)
+ operator = (const Tablero&t): Tablero&
+getDimX() :
+getDimY() : int
+getCasillaAt(const Coordenada&c): Casilla&
+operator () (char l, int y): Casilla&
<<friend>> operator << (ostream&o, const Tablero&t): ostream&
+colocaPiezaAt(const Coordenada &c, Pieza& p) : bool

9. FEB06

(1.5 punto) Dada la siguiente jerarquía de herencia, indicad la visibilidad de los atributos de la clase ProductoBancario en las clases CuentaJoven e Hipoteca.



Ampliación: El contador ProductoBancario::numProductos, que indica el número total de productos bancarios que existen en el sistema, es público y puede ser modificado por cualquier usuario de la jerarquía de clases. Indica cómo evitar ésto y en que lugares se debería inicializar, incrementar y decrementar dicho contador.

10. La empresa Objectic, para la cual trabajas, te ha pedido que implementes, en C++, una clase llamada TEntero que encapsule un dato de tipo entero y permita operar los objetos de dicha clase con el tipo básico entero (int) utilizando operadores aritméticos, de asignación, o poder utilizarlo con el operador de salida (<<). Sólo hay un problema: tu jefe te ha prohibido sobrecargar los operador aritmético y el de asignación, así como utilizar funciones amigas. Como ejemplo, el código siguiente debe compilar y funcionar sin errores ni advertencias del compilador:

```
int main() {
  int a=10,b=9;
  TEntero c = 2;
  TEntero d;

  cout << c+a+c << endl; // 14
  cout << c-a-c << endl; // -10
  cout << a+c*b-b*c+b/c-c/b; // 14
  d = a = c = 3;
  cout << d << c << a << endl; // 333
}
```

---

11. DIC06 Dada esta clase en C++:

```
class Mini {
  friend int operator+(Mini m1, Mini m2) { return
m1.v+m2.v; }
  int v;
  public:
    Mini(int val) : v(val) {}
    operator int() const { return v; }
};
```

Indica si existe algún tipo de ambigüedad en la sobrecarga del operador '+'. Ilustra tu respuesta con un ejemplo.

---

12. SEP06 Define una función genérica llamada 'Intercambio' que permita intercambiar el valor de dos objetos del mismo tipo. Indica qué restricciones debe cumplir el tipo de los objetos para poder usar la función Intercambio con ellos.

Ejemplo:

```
  Carta As1(As, Corazones);
  Carta As2(As, Treboles);

  Intercambio(As1, As2);
  cout << As1 << endl; // Imprime un As de Treboles
  cout << As2 << endl; // Imprime un As de Corazones
```

---

13. SEP06 Dadas la siguiente declaración de métodos

```
class T {
...
 public:
  T& operator++();
  T  operator++(int);
};
```

1. Indica cuál es la sobrecarga del operador de preincremento y cual es la del operador de postincremento.
2. ¿Cuál es el cometido del argumento de tipo entero?
3. ¿Por qué la primera versión devuelve una referencia y la segunda no?

---

14. DIC03
(1,5 puntos) Define una plantilla **resta**() que permita restar dos objetos de cualquier tipo, indicando que requisitos deben cumplir dichos objetos para que la plantilla funcione correctamente.

---

15. SEP07 (4 puntos) En el siguiente programa en C++, se define una clase genérica 'Vector' que permite crear vectores de objetos de cualquier tipo, añadir elementos al vector, obtener el elemento en una determinada posición y obtener el número de elementos que contiene el vector y su capacidad máxima. A la vista del programa principal, resulta evidente que pueden producirse ciertos errores en tiempo de ejecución que preferiríamos poder controlar mediante excepciones. **Se pide:**

a) Cuando se intente añadir un objeto a un vector que ya está lleno, debe lanzarse una excepción que, al ser capturada, indique la capacidad máxima del vector y que ésta ha sido superada.

b) Cuando se intente acceder a una posición (con el operador []) que no contiene ningún objeto previamente añadido con el método 'addElement', se debe producir una excepción que, al ser capturada, permita conocer la posición a la que se intentó acceder y el número máximo de posiciones válidas.

c) Se debe capturar en el programa principal la excepción bad_alloc

d) En el programa principal se debe capturar cualquier otra excepción imprevista, y emitir un mensaje de error genérico.

Nota: Para los puntos a y b, es aconsejable que crees tus propias clases de excepciones. Los objetos que se lanzan como excepción deben contener la información sobre el acceso incorrecto. Además, debes añadir tanto el código que lanza las excepciones (en la clase 'Vector') como el código que las captura (en el programa principal).

```
#include <iostream>
#include <string>

using namespace std;

template <typename T, long tam=10>
class Vector {
  T* array;
  int nelem;
```

```
  public:
    Vector() : nelem(0) { array = new T[tam]; }
    ~Vector() { delete [] array; array=NULL; nelem=0; }

    void addElement(const T& elem)  {
      if (nelem<tam) array[nelem++]=elem;
      /* else ¿? */
    }

    T operator[](int n) const {
      if (n<nelem) return array[n];
      /* else ¿? */
    }
    int getNumElem() const { return nelem; }
    int getCapacidad() const { return tam; }
};

int main() {
  Vector<string> cadenas;
  string s;
  int i=0;

  do {
    cin >> s;
    if (s!="0")
      cadenas.addElement(s);
  } while (s!="0");

  for (int j=cadenas.getCapacidad(); j>=0 ; j++)
    cout << cadenas[j];
}
```

---

16. DIC07 (4 puntos) Dado el siguiente código en C++, que no contiene ningún error de compilación:

```
#include <iostream>
#include <exception>

using namespace std;

  class EPilaVacia : public exception {
      public:
        const char* what() const throw()
          { return "Error: Pila Vacía"; }
  };

  template <class T, int max=100>
  class Pila {
    public:
      Pila() : cont(0) { pila = new T[max]; }
      virtual void apilar(T* pt) { if(cont<max) pila[cont++]=*pt; }
      virtual void apilar(T t) { if (cont<max) pila[cont++]=t; }
      T tope() const throw(EPilaVacia)
        { if (cont>0) return pila[cont-1]; else throw EPilaVacia(); }
      T& desapilar() throw (EPilaVacia)
        { if (cont>0) return pila[--cont]; else throw EPilaVacia(); }
      int size() const { return cont; }
    private:
      T* pila;
```

```
    int cont;
};

class Figura {
  public: virtual void print() { cout << " FIGURA " << endl; } };
class Circulo : public Figura {
  public: void print() { cout << " CIRCULO " << endl; } };
class Triangulo : public Figura {
  public: void print() { cout << " TRIANGULO " << endl; } };
```

contesta a los siguientes apartados:

a) (1.5 puntos) Completa la forma canónica ortodoxa de la plantilla Pila (tanto la declaración como definición de los métodos).

b) Dado el siguiente programa principal, consecutivo al código anterior:

```
1  int main() {
2
3     Circulo c1;
4     Pila<Circulo,50> circulos50;
5     Pila<Circulo>& rpc = circulos50;
6        for (int i=0; i<100; i++)
7          rpc.apilar(c1);
8
9     Pila<Triangulo*> triangulos;
10    Pila<Figura*>& rpf = triangulos;
11       rpf.apilar(new Triangulo());
12
13    Pila<Figura*> pfiguras;
14       pfiguras.apilar(new Circulo());
15       pfiguras.apilar(new Triangulo());
16
17    Pila<Figura*> pfiguras2(pfiguras);
18
19       pfiguras.tope()->print();
20       pfiguras.desapilar();
21       pfiguras.tope()->print();
22       pfiguras.desapilar();
23       pfiguras.~Pila();
24
25       pfiguras2.tope()->print();
26 }
```

b1) (1 punto) Indica los posibles errores de compilación que pueda contener, indicando en qué línea se producen y por qué causa.

b2) (0.5 puntos) Añade el código necesario para capturar la excepción EPilaVacia. (No es necesario que reescribas todo el código; indica únicamente el nuevo código y en que línea(s) lo insertarás.

b3) (1 punto) Indica la salida que produce el programa si lo ejecutamos tras juntar todo el código mostrado en un sólo fichero, eliminar las líneas con errores y compilarlo.