

# Seminario2.pdf



ersosio



Programación Orientada a Objetos



2º Grado en Ingeniería Informática



Escuela Superior de Ingeniería  
Universidad de Cádiz



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.





## 5 ECO TIPS FOREVER GREEN

SIGUE LOS CONSEJOS DE FOREVER GREEN  
PARA CUIDAR EL MEDIO AMBIENTE

1

### REUTILIZA

y recicla  
todo lo que puedas.

utilizar siempre objetos  
que puedan tener una  
segunda vida

2

### ENCIENDE

Las luces justas.

para ahorrar utiliza  
los horarios más baratos:  
- **llano** de 8:00 a 10:00 h,  
de 14:00 a 18:00 horas y  
de 22:00 a 00:00 horas  
- **valle** de 00:00 a 08:00 h.

3

### APROVECHA

todo el agua.

no desperdigies los  
primeros litros esperando  
a que salga caliente.



4

### EVITA

contaminar el  
entorno.

no depositar en el medio  
ambiente materiales o  
productos que son espe-  
cialmente agresivos para  
la salud y la naturaleza.

5

### USA BICICLETA

transporte público o  
comparte vehículo  
cuando lo utilices.  
Así se reduce el CO2

¿AÚN NO  
NOS CONOCES?



# Seminario Tema 2

## Tarea 2.1. Cuestiones

**Ejercicio 1**

¿Cómo implementa el lenguaje de programación C++ el principio de encapsulamiento?

Mediante clases, que agrupan datos y operaciones que se pueden realizar sobre ellas.



Programación Orientada a Objetos    2.1. Estudio de la estructura de clase

## Tarea 2.1. Cuestiones

**Ejercicio 2**

¿Cómo implementa el lenguaje de programación C++ el principio de ocultación de información?

Separando en secciones públicas y privadas los contenidos de la clase, a las secciones públicas tendrán acceso el código usuario y las secciones privadas ocultan los detalles de representación interna de la clase.



Programación Orientada a Objetos    2.1. Estudio de la estructura de clase

## Tarea 2.1. Cuestiones

### Ejercicio 3

¿Hay algún error en el siguiente programa? Si es así, explique por qué y corríjalo.

```
1 #include <iostream>
2 class C {
3 public:
4     C(int i = 0): n(i) {}
5     void mostrar() { std::cout << "i=" << n << std::endl; }
6     private: int n; 
```

El error está aquí  
↓

Explicación: Estamos declarando un objeto const C, la función C::mostrar() no modifica nada de ese objeto y no está marcada como const.

```
7     Solución: const
8 };
9
10 int main() { const C c; c.mostrar(); } 
```

Tip: Si el objeto estuviera declarado como 'C' y no como 'const C', no habría error.

## Tarea 2.1. Cuestiones

### Ejercicio 4

Indique los errores que hay en el siguiente código y su causa.

```
1 class C {
2 public:
3     C();
4     C(int a, int b, int c, int d);
5     int f1(int i) const;
6     int f2(int i);
7     static void f3() {m = 1;} 
```

Al existir una m para todos los objetos de la clase C y al existir una l para cada uno de los objetos de la clase, además las funciones static sólo pueden trabajar con miembros static de la clase ya que no recibe \*this.

```
8     static int n;
9     private:
10    mutable int i;
11    const int j;
12    mutable int k;
13    int l;
14    static int m; 
```

→ La palabra 'static' en el entorno de clase convierte la variable en miembro de la clase, no de objeto. Solo existe un m para todos los objetos de la clase.

```
15 }; 
```

Tip: static también lo podemos usar como clase de almacenamiento.

## Tarea 2.1. Cuestiones

### Ejercicio 4 (cont.)

```
17 int C::f1(int i) const
18 {
19     l = i; k = i; return 0;
20 }
21 C::C() {i = j = k = l = 0;}
22 C::C(int a, int b, int c, int d) : i(a), j(b), k(c), l(d) {}

26 int C::f2(int i)
27 {
28     k = i; l = i; return 0;
29 }
```

→ Es un método const, pero modifica una variable de tipo int ( $l = i$ ). Con  $k = i$  no habría problemas ya que  $K$  es un miembro mutable.

→ La asignación a  $j$  es imposible ya que es una variable de tipo const, para inicializar a  $j$  usamos la lista inicializadora.

Programación Orientada a Objetos 2.1. Estudio de la estructura de clase

## Tarea 2.1. Cuestiones

### Ejercicio 5

Considerando el código de la pregunta anterior, determine si hay errores, en tal caso corríjalos, e indique qué imprime el programa.

```
1 #include <iostream>
2 class C { // clase C anterior
3     ...
4 };
5 // métodos de la clase C
6 // aquí ...
7
8 int main()
9 {
10     C c;
11     C::n = 3;
12     c.n = 4; ← NO RECOMENDADO
13     std::cout << C::n << "\u201c" << c.n << std::endl;
14     return 0;
15 }
```

→ Debe ser inicializado fuera del main, las inicializaciones de los miembros estáticos han de hacerse fuera de la clase y fuera del main. `int C::n = 3;`

↑ Para inicializar

Programación Orientada a Objetos 2.1. Estudio de la estructura de clase

## Tarea 2.1. Cuestiones

### Ejercicio 6

Considere la siguiente clase:

```
1 class C {  
2 public:  
3     C(int i) : n(i), m(0.0) {} // ctor. de conversión  
4     // ...  
5 private: miembro: C operator +(const C& c)  
6     int n; externa: C operator +(const C& c1, const C& c2)  
7     double m;  
8 };
```

Escriba las declaraciones correspondientes a la sobrecarga del operador + para objetos de tipo C como miembro y como función externa. A continuación escriba a modo de ejemplo un trozo de código que provoque un error de compilación con el operador miembro, pero no con el externo.

## Tarea 2.2. Cuestiones

### Ejercicio 1

¿Se puede definir un constructor con un nombre diferente al de la clase? Justifique la respuesta.

No, las reglas sintácticas del lenguaje especifican que el constructor ha de tener el mismo nombre que la clase.

Además, a los constructores se llaman implícitamente mientras que a los métodos hay que llamarlos explícitamente.

## Tarea 2.2. Cuestiones

1. Los miembros constantes puedes inicializarlos, pero no puedes asignarles ningún valor. Ocurre lo mismo con los miembros de tipo de referencia.
2. Como su propio nombre dice, las listas de inicialización inicializan el/los miembro/s de la clase

### Ejercicio 2

Enumere las diferencias existentes entre inicializar un atributo en la lista de inicialización y asignarle un valor en el cuerpo del constructor. ¿Es posible utilizar dicha lista en otros métodos de una clase? *No es posible, las reglas del lenguaje lo prohíben, ya que los métodos no inicializan objetos.*



## Tarea 2.2. Cuestiones

### Ejercicio 3

```
1 class punto {  
2     double x, y;  
3 public:  
4     punto(double a = 0., double b = 0.) : x{a}, y{b} {}  
5     punto(const punto &p) : x{p.x}, y{p.y} {}  
6     punto& operator =(const punto &p)  
7         { x = p.x; y = p.y; return *this; }  
8 };
```

Diga qué función de la clase punto se llama en cada una de las siguientes líneas, o si es incorrecta, suponiendo que se han ejecutado las anteriores, corregidas si es necesario.

- 1 punto p; *Línea 4* Correcta, constructor por omisión  
2 punto q(); *Línea 4* Correcta, es una declaración de función  
2. = 2.0 3 punto r(2.); *Línea 4* Error, sobra la coma o pones un punto v = r; Correcto, llamamos al cto por omisión  
4 punto s{3.4}; *Línea 4* Correcto, constructor de conversión  
5 punto t{}; *Línea 4* Correcto, llamamos al cto por omisión  
6 punto u(q); *Línea 4* Error, q es una función  
7 punto v = r; *Línea 4* Correcto, llamamos al cto por copia ya que inicializa  
8 t = s; *Línea 4* Correcto, llamamos al operador de asignación.

## Tarea 2.2. Cuestiones

### Ejercicio 4

Dadas las clases Libro1 y Libro2

```
1 class Libro1 {           8 class Libro2 {  
2     string titulo_; int pags_;    9     string titulo_; int pags_;  
3 public:                 10 public:  
4     Libro1(string t="",        11     Libro2(string t, int p=0);  
5         int p=0);            12     Libro2(const char* c);  
6     // ...                  13     // ...  
7 };                      14 };
```

decida si x se puede sustituir por 1 o 2 en los siguientes items:

- ① Se puede definir: LibroX lib1; *Sólo Libro1 ya que es la única clase con cto por omisión*
- ② Se tiene un constructor de conversión de std::string a LibroX *En ambas clases se puede ya que en ambas clases tiene un cto por conversión, omitiendo el .1 parte.*
- ③ Se puede definir: LibroX lib2[5]; *Sólo Libro1 por la misma razón que el 1.*

## Tarea 2.2. Cuestiones

### Ejercicio 4 (cont.)

```
1 class Libro1 {           8 class Libro2 {  
2     string titulo_; int pags_;    9     string titulo_; int pags_;  
3 public:                 10 public:  
4     Libro1(string t="",        11     Libro2(string t, int p=0);  
5         int p=0);            12     Libro2(const char* c);  
6     // ...                  13     // ...  
7 };                      14 };
```

- ④ Se puede definir: std::vector<LibroX> lib3; *En ninguna*
- ⑤ Se produce una conversión implícita de const char\* a string al ejecutar LibroX\* lib4 = new LibroX("El Quijote"); *Sólo Libro2*
- ⑥ Se puede definir: LibroX lib5 = "El Quijote"; *Sólo en Libro2*

## Tarea 2.2. Cuestiones

### Ejercicio 5

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;

5 class Libro {
6     char* titulo_; int paginas_;
7 public:
8     Libro() : titulo_(new char[1]), paginas_(0) {*titulo_ = 0;}
9     Libro(const char* t, int p) : paginas_(p) {
10         titulo_ = new char[strlen(t) + 1];
11         strcpy(titulo_, t);
12     }
13     ~Libro() { delete[] titulo_; }
14     void paginas(int p) { paginas_ = p; }
15     int paginas() const { return paginas_; }
16     char* titulo() const { return titulo_; }
17 };
```

Programación Orientada a Objetos 2.2. Constructores y uso de objetos

## Tarea 2.2. Cuestiones

### Ejercicio 5 (cont.)

```
18 void mostrar(Libro l) {
19     cout << l.titulo() << " tiene "
20             << l.paginas() << " páginas" << endl;
21 }
22 int main() {
23     Libro l1("Fundamentos de C++", 474),
24         l2("Por Fin: C ISO", 224),
25         l3;
26
27     l3 = l1;
28     mostrar(l1), mostrar(l2), mostrar(l3);
29 }
```

El compilador no nos daría ningún error, ya que usaría el operador = definido por el compilador, pero cabe a destacar que no funcionaría bien ya que tras la línea 29 Diga si el programa funciona correctamente. En caso afirmativo indique lo que imprime. En caso negativo haga las modificaciones necesarias para que funcione correctamente.

\*27, l3 y l1 comparten el mismo puntero a memoria, por lo que si realizamos un cambio a la cadena de l3, l1 también se vería afectada.

## Tarea 2.2. Cuestiones

### Ejercicio 6

¿Se consigue llamar a la función f o se producirá un error? En tal caso, corríjalo.

```
1 struct B;
2 struct A {
3     A(B);
4     // ...
5 };
6 struct B {
7     operator A();
8     // ...
9 };
```

Se produce un error de ambigüedad.

El procedimiento f espera un objeto A, pero en el main se le manda un objeto B. El compilador intentará realizar una conversión implícita de B a A, pero se encuentra con 2 funciones:

- El cto de conversión de la clase A.
- El operador A() de la clase B.

Ante esto, el compilador muestra un error de ambigüedad.

```
7     operator A();      Ante esto, el compilador muestra un error de ambi-  
8     //...  
9 };  
  
11 void f(A&);  
  
13 main() { B b; f(b); }
```

## Tarea 2.3. Cuestiones

## Ejercicio 1

Clasifique las funciones y operadores miembro de la clase `matriz` en diferentes categorías (**constructores**, **destructores**, **observadores** y **modificadores**). 

→ No tiene, usa el destructor predeterminado del compilador.

## matriz.h

```
1 // Clase para matrices de números en coma flotante y doble precisión.

3 #ifndef MATRIZ_H_
4 #define MATRIZ_H_
5 #include <valarray>
6 #include <initializer_list>

8 using std::valarray;
9 using std::slice;
10 using std::slice_array;
11 using std::initializer_list;

13 class matriz {
14 public:
15     // Constructores
16     explicit matriz(size_t m = 1, size_t n = 1, double y = 0.0);
17     matriz(size_t m, size_t n, double f(size_t i, size_t j));
18     matriz(const initializer_list<valarray<double>>& l); // C++11

ejerci
AQUÍ HAY 4 constructores
```

## matriz.h

```
20 // Copia y movimiento (C++11)
21 matriz(const matriz&) = default;
22 matriz(matriz&&) = default;
23 matriz& operator =(const matriz&) = default;
24 matriz& operator =(matriz&&) = default;

26 // Dimensión
27 size_t filas() const;
28 size_t columnas() const;

30 // Operadores de indexación
31 double operator ()(size_t i, size_t j) const;
32 double& operator ()(size_t i, size_t j);
33 valarray<double> operator []()(size_t i) const;
34 slice_array<double> operator []()(size_t i);
35 valarray<double> operator ()(size_t j) const;
36 slice_array<double> operator ()(size_t j);
```

Programación Orientada a Objetos Un ejemplo: La clase matriz

## matriz.h

```
38 // Asignación de valor constante
39 matriz& operator =(double y);

41 // Operadores de asignación compuesta
42 matriz& operator +=(const matriz& a);
43 matriz& operator -=(const matriz& a);
44 matriz& operator *=(const matriz& a);
45 matriz& operator *=(double y);

47 // Matriz opuesta
48 friend matriz operator -(const matriz& a); → No es un elbo
de la clase
50 private:
51     size_t m, n;
52     valarray<double> x;
53 };
```

Programación Orientada a Objetos Un ejemplo: La clase matriz

## matriz.h

```
55 // Prototipos de operadores externos no amigos que trabajan con matrices.  
56 const matriz& operator +(const matriz& a);  
57 matriz operator +(const matriz& a, const matriz& b);  
58 matriz operator -(const matriz& a, const matriz& b);  
59 matriz operator *(const matriz& a, const matriz& b);  
60 matriz operator *(double y, const matriz& a);  
61 matriz operator *(const matriz& a, double y);  
  
63 // Definiciones en linea.  
64 #include "matriz-inline.h"  
  
66 #endif // MATRIX_H_
```

Programación Orientada a Objetos Un ejemplo: La clase matriz

## Tarea 2.3. Cuestiones

### Ejercicio 2

Si existen los siguientes constructores, escriba una instrucción en cada caso en la que se invoque al mismo y diga si se utiliza en el programa de prueba (en caso afirmativo indique dónde).

- Constructor predeterminado ● matriz m;
- Constructor de copia ● matriz b = m;
- Constructor de movimiento ● matriz a (std::move(b));
- Constructor de conversión ● matriz n (2);

Programación Orientada a Objetos 2.3. La clase matriz

## Tarea 2.3. Cuestiones

### Ejercicio 3

- Describa los errores que hay en el siguiente código:

```
1 matriz A = 3; Al ser explicit, no podemos hacer eso
2 matriz B = matriz(5);
3 matriz C(3);
4 B = 2;
5 A = matriz(5);
6 B = static_cast<matriz>(4);
```

- ¿Por qué se declara `explicit` el primer constructor de la clase `matriz`? ¿Podría causar algún problema si no se hiciera así? Y en caso afirmativo, ¿se podría evitar ese problema definiendo un operador de conversión de `int` a `matriz`? Razone la respuesta.

Se pone `explicit` para no permitir que el compilador lo llame de forma implícita al intentar hacer conversiones implícitas. Si quitamos el `explicit` el compilador no sabría transformar de forma `int` a `matriz` o de `int` a `double`. Ese operador sería equivalente al cto de conversión, pero sin el `explicit`, daría error de ambigüedad.

## Tarea 2.3. Cuestiones

### Ejercicio 4

¿Por qué devuelven tipos distintos los operadores de signo + y -?

El operador de signo - devuelve un nuevo objeto matriz.

El operador de signo + devuelve directamente el operador.

## Tarea 2.3. Cuestiones

### Ejercicio 5

¿El operador - de cambio de signo es miembro de la clase matriz?

¿Se podría definir de la otra forma? En caso afirmativo, escriba la declaración. ¿Qué ventajas e inconvenientes tendría?

No es miembro de la clase.

Sí, matriz& operator - ();

Programación Orientada a Objetos 2.3. La clase matriz

## Tarea 2.3. Cuestiones

### Ejercicio 6

¿Es correcto definir el operador \*= como sigue?

```
1 inline matriz& matriz::operator *=(const matriz& a)
2 {
3     n = a.columnas();
4     x *= a.x;
5     return *this;
6 }
```

No está correcto, ese operador \*= es de la clase valarray  
y realiza la multiplicación como si fuera un vector y  
no elto a elto.

Programación Orientada a Objetos 2.3. La clase matriz