SEMINARIO C++

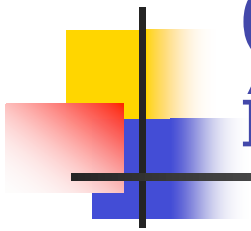
Introducción a la Programación Orientada a Objetos

Parte 2
v. 20100921

Pedro J. Ponce de León

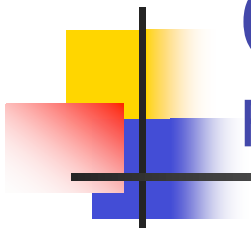
Depto. Lenguajes y Sistemas Informáticos - Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante



C++
ÍNDICE

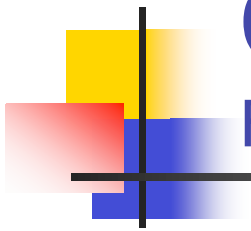
1. **Operaciones set/get/is.**
2. El puntero *this*
3. Los constructores
4. El destructor
5. Doxygen
6. Utilidad de compresión tar
7. UML
8. Práctica 0 v. 2.0
9. Entrega de prácticas en web DLSI



C++
FUNCIONES O MÉTODOS SET/ GET / IS.

- Por el principio de encapsulación, casi nunca es conveniente acceder directamente a los atributos de una clase. Lo usual es definirlos como **private** y, para acceder a ellos, implementar funciones **set/get/is** (llamadas tb **ACCESORES**): `getDia(), setMes(), isBisiesto(),…`

- Los métodos **get** permiten representar la información dentro de la clase de una forma y proporcionarla de una forma distinta.
  - Por ejemplo, el DNI de una persona puede representarse internamente como un atributo de tipo long y otro de tipo char, pero el método getDNI() podría devolver el DNI como una cadena de caracteres.



C++
FUNCIONES O MÉTODOS SET/ GET / IS.

- Los métodos **set** permiten hacer algo más que simplemente asignar un valor a un atributo (por ejemplo, validar dicho valor) de forma transparente al usuario de la clase.

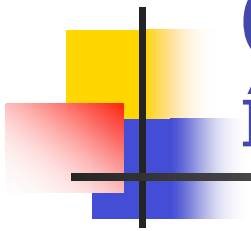
```
void setEdad(int laEdad) {
  if (laEdad<0) edad=0;
  else edad=laEdad;
}
```

- Por otro lado, para consultar si un atributo cumple una determinada propiedad, por convenio utilizamos funciones de tipo **is**
  - P. ej. `isBisiesto();`

C++
FUNCIONES O MÉTODOS SET/ GET.

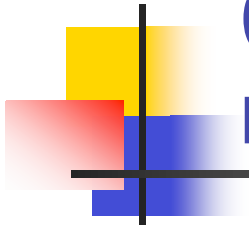
```
class Fecha {
public:
   void setDia(int d) {dia =d;};
   void setMes(int m) { mes = m; }
   void setAnyo(int a) { anyo = a; }
   int getDia() { return dia; }
   int getMes() { return mes; }
   int getAnyo() { return anyo; }
   bool isBisiesto()
     {return ((anyo%4)==0);}
private:
   int dia, mes, anyo;
};
```

Fecha
- int dia - int mes - int anyo;
+void setDia(int d) + void setMes(int m) + void setAnyo(int a) + int getDia() + int getMes() + int getAnyo() + bool isBisiesto();



C++
ÍNDICE

1. Operaciones set/get/is.
2. **El puntero *this***
3. Los constructores
4. El destructor
5. Doxygen
6. Utilidad de compresión tar
7. UML
8. Práctica 0 v. 2.0
9. Entrega de prácticas en web DLSI



C++
EL PUNTERO THIS

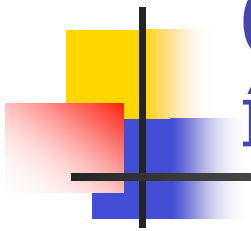
- El puntero **`this`** es una pseudovariable
  - No se declara
  - No se puede modificar
- En C++ es un **argumento implícito** que reciben las funciones miembro

- Es un puntero al objeto a través del cual se invoca un método (llamado objeto 'receptor').

- Es necesario cuando
  - Queremos desambiguar nombre de parámetro y nombre de dato miembro

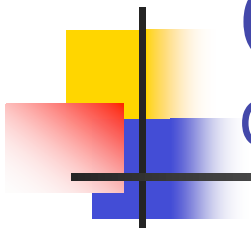
```
void Fecha::setMes(int mes){
  // mes=mes; ->ERROR: ambiguo
  this->mes=mes;
}
```

  - Queremos obtener la dirección de memoria del objeto receptor.



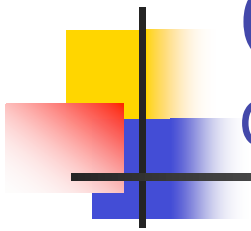
C++
ÍNDICE

1. Operaciones set/get/is.
2. El puntero *this*
3. **Los constructores**
4. El destructor
5. Doxygen
6. Utilidad de compresión tar
7. UML
8. Práctica 0 v. 2.0
9. Entrega de prácticas en web DLSI



C++
CONSTRUCTOR de una clase

- **Constructor**: función miembro de la clase cuyo objetivo es construir objetos e inicializarlos.
  - Tiene el mismo nombre que la clase.
  - NO devuelve valores (ni siquiera void).
  - Puede admitir parámetros como otra función.
  - Suele estar en la parte pública.
  - El constructor es invocado implícitamente al crear un objeto.
  - Si no se define, el compilador genera uno (a qué valores inicialice los datos miembro depende del compilador)
  - Es conveniente definir siempre un constructor sin parámetros (**CONSTRUCTOR POR DEFECTO)**

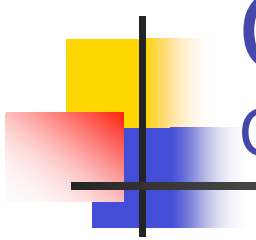


C++
CONSTRUCTOR POR DEFECTO

```
class Fecha
{
  public:// Parte pública de la clase
    Fecha( );  //Constructor por defecto
    Fecha(int,int,int); //Constructor sobrecargado
    Fecha(const Fecha&); //Constructor de copia
...
};

Fecha::Fecha()
{ // Inicializamos la fecha a 01/01/1900.
  dia = 1;    //También podríamos poner setDia(1);
  mes = 1;    //También podríamos poner setMes(1);
  anyo = 1900;       //También setAnyo(1900);
}
```

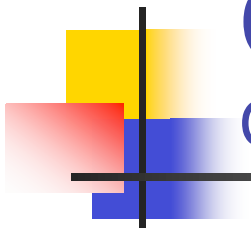
Constructor por defecto



C++
CONSTRUCTOR SOBRECARGADO

```
Fecha::Fecha(int d, int m, int a)
{
  dia = d;  //O bien, setDia(d);
  mes = m;  //O bien, setMes(m);
  anyo = a; //O bien, setAnyo(a);
}
```

Constructor sobrecargado (con argumentos)

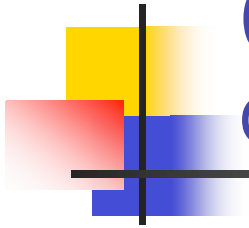


C++
CONSTRUCTOR: **lista de inicialización**

Sección de código en los constructores que permite inicializar datos miembro antes de ejecutar el cuerpo del constructor.

```
Fecha::Fecha(int d, int m, int a)
: dia(d), mes(m), anyo(a)
{ }
```

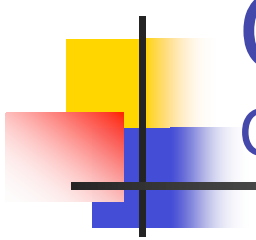
Lista de inicialización



C++
CONSTRUCTO: lista de inicialización

- La lista de inicialización es más eficiente que realizar asignaciones en el cuerpo del constructor
  - En la lista de inicialización, se invoca directamente a los constructores de los atributos.
  - Si usamos asignación, todos los atributos se construyen con su constructor por defecto antes de entrar en el cuerpo del constructor y luego se invoca al operador de asignación.

```
Fecha::Fecha(int d, int m, int a)
: dia(d), mes(m), anyo(a)
{ }
```

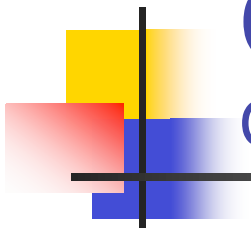


C++
CONSTRUCTOR SOBRECARGADO: **valores por defecto**

- Cualquier método (constructores incluidos) puede tener valores por defecto para sus argumentos.

```
class Fecha {
…
  public:
    Fecha(int d=1, int m=1, int a=1970);
    …
…};

Fecha::Fecha(int d, int m, int a)
// los valores por defecto se indican sólo en la declaración
{
  dia = d;      //O bien, setDia(d);
  mes = m;      //O bien, setMes(m);
  anyo = a;     //O bien, setAnyo(a);
}
```



C++
CONSTRUCTOR DE COPIA

- Crea un nuevo objeto a partir de otro del mismo tipo.
  - Un solo argumento: una **referencia** constante a un objeto de la misma clase.

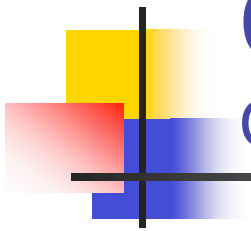
```
Fecha(const Fecha &);
Fecha c(d); Fecha c=d;
```

- Si no se proporciona uno, el compilador genera uno por defecto que hace una copia bit a bit:
  - *No válido si el objeto original tiene atributos en memoria dinámica*
- Es similar a la asignación, pero no igual: ni la sustituye ni implementa.

```
Fecha c=d;
// invoca a ctor. de copia (una operación)
```

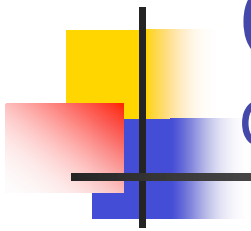
≠

```
Fecha c; c=d;
//ctor por defecto + asignación (2 operaciones)
```



C++
CONSTRUCTOR DE COPIA.

```
Fecha::Fecha(const Fecha &tf)
: dia(tf.dia), mes(tf.mes), anyo(tf.anyo)
{}
```



C++
CONSTRUCTOR DE COPIA: Invocación implícita

- Se invoca automáticamente cuando:
  - Se inicializa explícitamente un objeto a partir de otro:

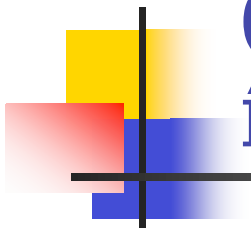
```
Fecha c(d); Fecha c=d;
```

  - Al pasar un argumento ***por valor*** a una función:

```
Fecha SumaFechas(Fecha p);
```

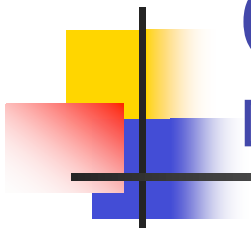
  - Al devolver una función un resultado **por valor** que es objeto de esa clase:

```
Fecha QueDiaEsHoy() { Fecha f; ...return (f);}
```



C++
ÍNDICE

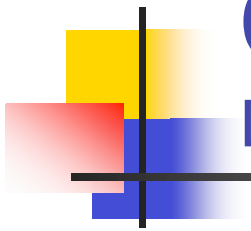
1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. **El destructor**
6. Doxygen
7. Utilidad de compresión tar
8. UML
9. Práctica 0 v. 2.0
10. Entrega de prácticas en web DLSI



C++
DESTRUCTOR

- Realiza la operación opuesta de un constructor, limpiando el almacenamiento asignado a los objetos cuando se crean.
- Tiene el mismo nombre que la clase, pero precedido por el símbolo **~**.
- No recibe ningún argumento ni devuelve ningún tipo de dato (ni void)

```
class Fecha {
  public:
    …
     ~Fecha() { dia=-1; mes=-1; anyo=-1; }
…};
```



C++
DESTRUCTOR

- El compilador llama automáticamente a un destructor del objeto cuando el objeto sale fuera del ámbito.

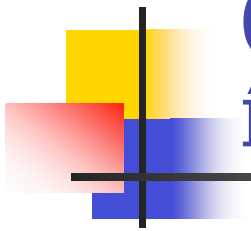
```
int Suma() {Fecha a; ...}
```

- También se invoca al destructor al hacer **delete**:

```
int Suma() { Fecha *f = new Fecha();… delete f; }
```

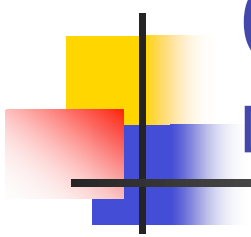
- Se puede invocar explícitamente pero no es aconsejable:

```
Fecha a; a.~Fecha();
```



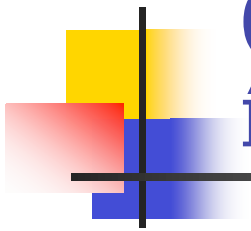
C++
ÍNDICE

1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. El destructor
6. **Doxygen**
7. Utilidad de compresión tar
8. UML
9. Práctica 0 v. 2.0
10. Entrega de prácticas en web DLSI



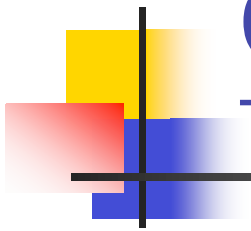
C++
DOXYGEN

- Herramienta para la generación de documentación (HTML entre otros formatos) a partir de comentarios en el código fuente.
- Descargad y consultad tutorial de Doxygen en la sección de materiales del C.V.



C++
ÍNDICE

1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. El destructor
6. Doxygen
7. **Utilidad de compresión tar**
8. UML
9. Práctica 0 v. 2.0
10. Entrega de prácticas en web DLSI



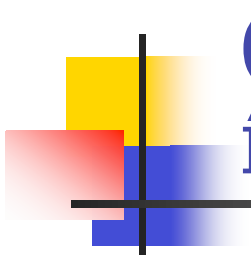
C++
TAR

- Usar comando *tar* (comprimir):

```
$ tar cvzf prac.tgz <dir. de
  trabajo>
```

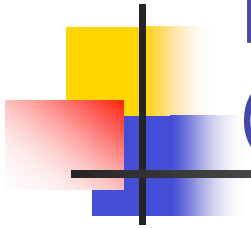
- Para descomprimirlo en la siguiente sesión:

```
$ tar xvzf prac.tgz
```



C++
ÍNDICE

1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. El destructor
6. Doxygen
7. Utilidad de compresión tar
8. **UML**
9. Práctica 0 v. 2.0
10. Entrega de prácticas en web DLSI

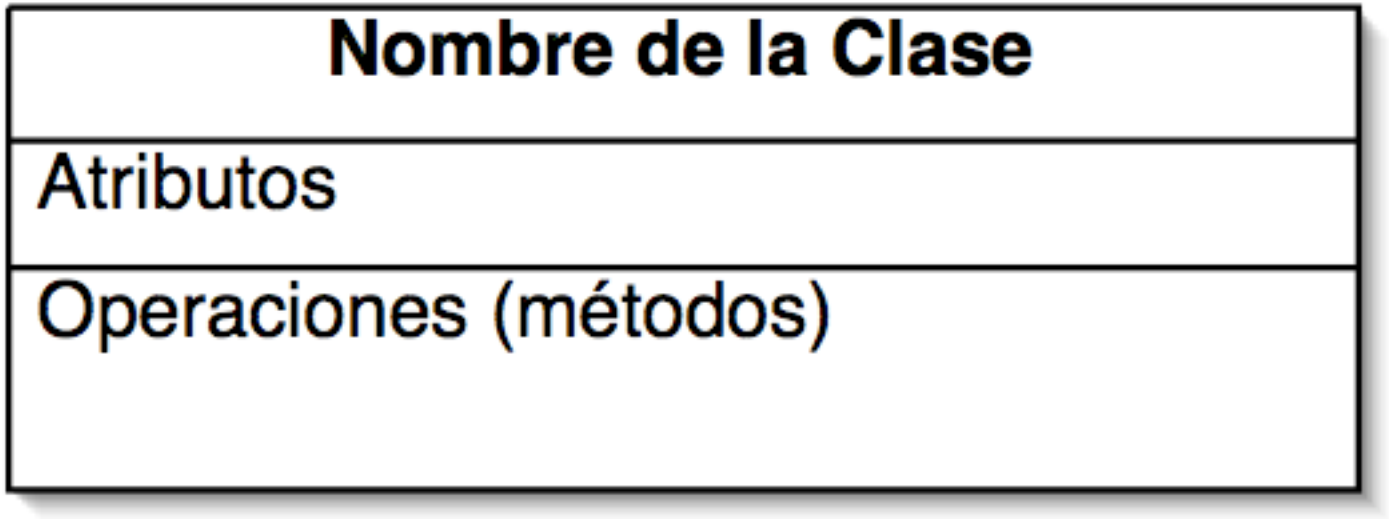


Breve introducción a UML
(Unified Modeling Language)

- UML es un lenguaje gráfico que se utiliza para el análisis y diseño de aplicaciones.
- Permite comunicar de forma precisa las ideas de diseño al equipo de desarrollo
- En POO usaremos únicamente el **diagrama de clases**
- Este diagrama especifica qué clases componen el sistema, sus atributos, su interfaz y sus relaciones con otras clases.
- En prácticas usaremos el diagrama de clases **detallado**

UML
Diagrama de clases

- Cada clase se representa por una caja con tres secciones:

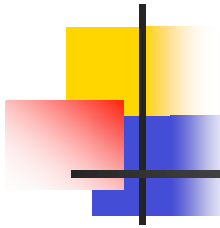


UML
Diagrama de clases

Especificación de atributos y operaciones:

NombreClase
nombreAtributo : tipoAtributo [= valorPorDefecto]
nombreOperacion(<lista de argumentos>) : tipoDevuelto

- El valor por defecto de un atributo es opcional
- La lista de argumentos de un método se especifica del mismo modo que los atributos, separando los argumentos por comas. Se puede omitir el nombre de los argumentos e indicar únicamente su tipo.



UML

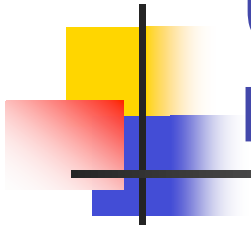
Diagrama de clases

Visibilidad pública (+) y privada (-):

NombreClase
+ atributoPublico : tipoX - atributoPrivado : tipoY
+ metodoPublico(<lista de argumentos>) : tipoDevuelto - metodoPrivado(<lista de argumentos>) : tipoDevuelto

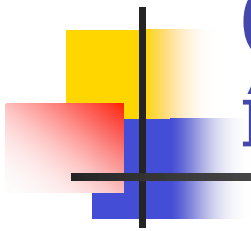
Métodos constantes:

NombreClase
+ <<const>> metodoConstante(<lista de argumentos>) : tipoDevuelto



UML
Diagrama de clases

- La especificación de relaciones entre objetos y entre clases, así como otras características de los diagramas de clases se tratará en sesiones posteriores.



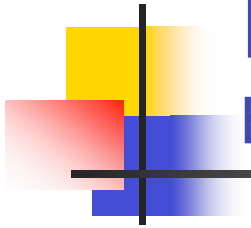
C++
ÍNDICE

1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. El destructor
6. Doxygen
7. Utilidad de compresión tar
8. UML
9. **Práctica 0 v. 2.0**
10. Entrega de prácticas en web DLSI

PRÁCTICA 0 v. 2.0

ENUNCIADO

Punto
- m_longitud : double - m_latitud : double
+ Punto(longitud : double = 0.0, latitud : double = 0.0) + Punto(const Punto &) + ~Punto() + operator=(const Punto &) : Punto & + setLong(double) : bool + setLat(double) : bool + getLong() : double + getLat() : double + imprimir() : void



PRÁCTICA 0 v. 2.0

ENUNCIADO

- Dado el diagrama de clases de la figura anterior, se pide:
  - Modificad la implementación de la clase Punto de la sesión anterior. Se trata de incorporar los constructores, el destructor y el operador de asignación.
  - Documenta tu código utilizando Doxygen y genera la documentación en HTML.
  - Esta versión de la práctica es la que has de entregar en el servidor de prácticas del DLSI.
    - Recuerda: sólo debes entregar código fuente (haz 'make clean' antes de empaquetar tu código).

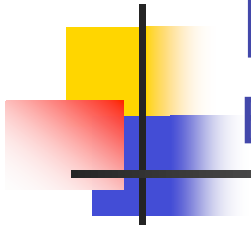
PRÁCTICA 0 v. 2.0
ENUNCIADO

- ***Sobrecarga del operador de asignación (=)***
  - La misión del operador de asignación es la misma que la del constructor de copia, excepto que trabaja sobre un objeto ya creado y debe devolver siempre una referencia al objeto receptor.

```
// definición 'inline'
Fecha& operator=(const Fecha& f) {
  if (this != &f)
  // protección contra autoasignación
  {    d=f.d; m=f.m; a=f.a; }
  return *this;
}
```

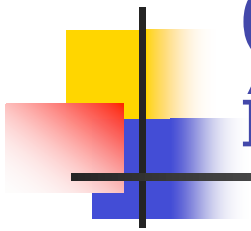
  - Se almacena el resultado de la operación en el objeto receptor
  - Se devuelve referencia al objeto receptor (esto permite concatenar asignaciones)

```
Fecha a,b,c;
a=b=c; // a.operator=(b.operator=(c));
```



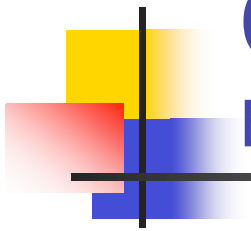
PRÁCTICA 0
EVALUACIÓN DE LA PRÁCTICA

- **Los requisitos imprescindibles para considerar correcta la práctica son:**
  - La práctica debe funcionar sin errores. En particular, no se debe producir ningún error del tipo “segmentation fault”, “null pointer assignment”, etc.
  - No se deben utilizar variables globales.
  - La práctica debe compilar correctamente con la orden **make** desde línea de comandos.



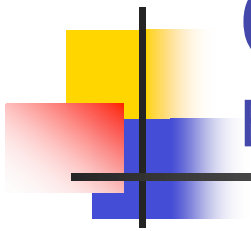
C++
ÍNDICE

1. Operaciones set/get/is.
2. Atributos de clase
3. El puntero *this*
4. Los constructores
5. El destructor
6. Doxygen
7. Utilidad de compresión tar
8. UML
9. Práctica 0 v. 2.0
10. **Entrega de prácticas en web DLSI**



C++
Entrega de prácticas en web DLSI

- Acceso con la misma cuenta (usuario y contraseña) que en el Campus Virtual:
  - Consulta de notas
  - Entrega de prácticas
  - Reserva de tutorías



C++
Entrega de prácticas en web DLSI

- El nombre del fichero a entregar debe ser

  p0-10-11.tgz

  Para que la entrega sea correcta, la práctica debe compilar correctamente con el *makefile* propuesto en el enunciado.
- Esta entrega de prueba no se evaluará.
- **¡AVISO!** Aquellos que tengan problemas en la entrega de la practica 1 por no haber probado antes el sistema con la práctica 0 se considerarán irrevocablemente suspendidos en esa práctica.

- Plazo: hasta el 19 de Octubre de 2009 a las 23:59h