SEMINARIO C++

Introducción a la Programación Orientada a Objetos con C++

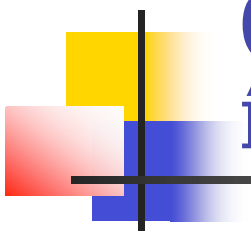
Parte I

v. 20100916

Pedro J. Ponce de León

Depto. Lenguajes y Sistemas Informáticos - Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

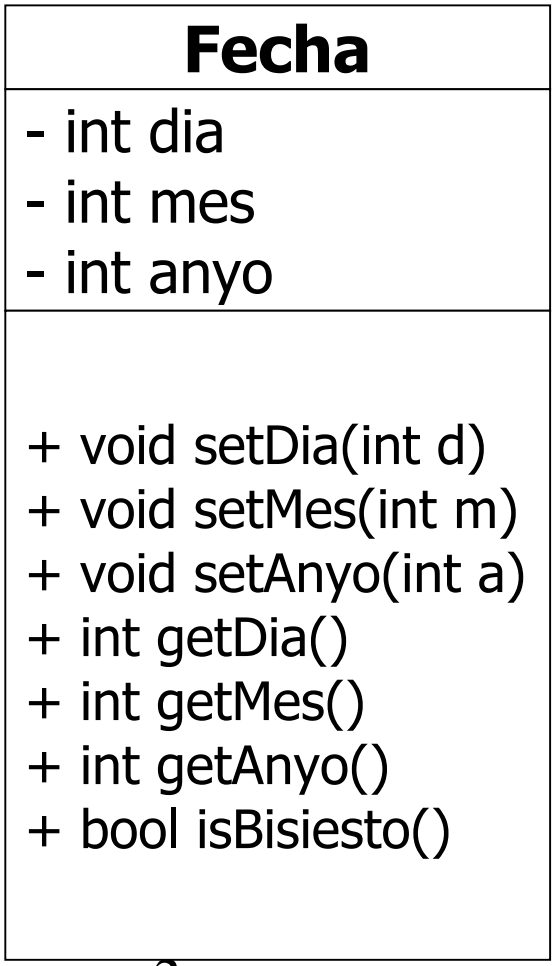


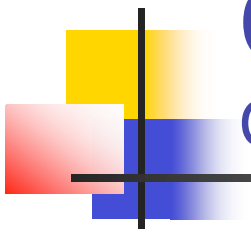
C++
ÍNDICE

1. **Clases en C++: estructura, ámbito de variables.**
2. Declaración y definición de clases
3. Instanciación de objetos
4. Estructura de un programa
5. Operaciones *inline* y constantes
6. Un ejemplo completo
7. Compilador: g++
8. Directivas de compilación
9. Make
10. Práctica 'cero' (1ª versión)

C++
CLASES : ESTRUCTURA

- Una clase es un tipo de dato (TAD) definido por el usuario.
- La clase define un conjunto de objetos que comparten las mismas propiedades: atributos y métodos.
  - **Atributos** (*datos miembro en C++*): información que cada uno de los objetos instanciados a partir de ella desea guardar de sí mismo
  - **Métodos** (*funciones miembro en C++*): instrumentos para la manipulación de atributos/relaciones

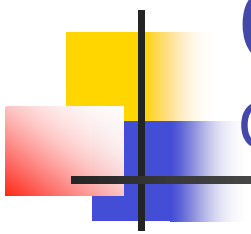




C++
CLASES : VISIBILIDAD (ÁMBITO)

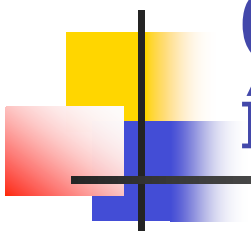
- **Modificadores de acceso**: especifican el ámbito desde el cual puede accederse a una propiedad.
  - **`private` (-)** : Sólo permite el acceso desde funciones miembro (métodos) de la clase.
  - **`public` (+)**: Permite el acceso desde cualquier lugar.

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()



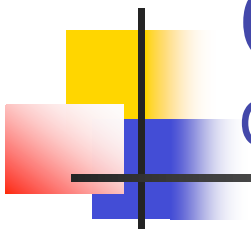
C++
CLASES : ENCAPSULACIÓN

- Siguiendo el principio de **encapsulación** de la POO, normalmente sólo la declaración de *operaciones* es **pública**, quedando oculta a los usuarios de la clase la forma en la que se han programado las distintas funciones miembro de la clase, así como los datos miembros.



C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. **Declaración y definición de clases**
3. Instanciación de objetos
4. Estructura de un programa
5. Operaciones *inline* y constantes
6. Un ejemplo completo
7. Compilador: g++
8. Directivas de compilación
9. Make
10. Práctica 'cero' (1ª versión)



C++
CLASES : **DECLARACIÓN**

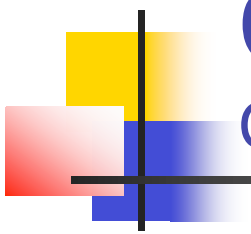
```
class <nombre_clase>
{
   public:
   ...
        Parte pública de la clase. Interfaz que ésta
   ofrece a los usuarios para que éstos manejen los
   datos.
   private:
   ...
        Parte privada de la clase. Incluye normalmente
   la declaración de atributos. Visibilidad por defecto
   de una clase (dif. respecto a struct).
};
```

C++
CLASES : **DECLARACIÓN**

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()

```
class Fecha
{
   public:
        void setDia(int d);
        int getDia();
        …
        bool isBisiesto();
   private:
        int dia;
        int mes;
        int anyo;
};
```

Fichero declaración (Fecha.h)



C++
CLASES : **DECLARACIÓN**

- **ORDEN DE DECLARACIONES EN UNA CLASE**

  - Los rasgos más importantes, normalmente en la parte publica, deberían estar listados al inicio de la declaración de la clase.

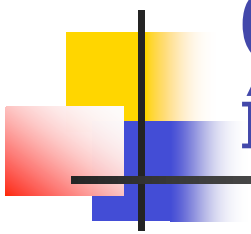
  - Los datos privados sólo son importantes para el desarrollador de la clase. Por tanto deberían estar listados hacia el final de la definición de dicha clase.

C++
CLASES : **DEFINICIÓN** (IMPLEMENTACIÓN)

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()

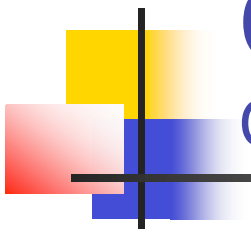
Fichero implementación (Fecha.cc)

```
#include "Fecha.h"

void Fecha::setDia(int d) {
  dia = d;
  // acceso a atributos:
  // directamente como si fuera var.
  // local a la función
}

bool Fecha::isBisiesto() {
  return (getAnyo() % 4) == 0;
  // acceso directo a métodos
}
… // '::' es el operador de
   resolución de ámbito
```



C++
ÍNDICE

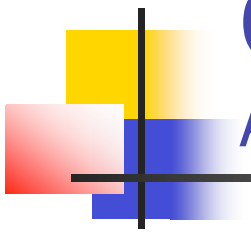
1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. **Instanciación de objetos**
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. Compilador: g++
9. Directivas de compilación
10. Make
11. Práctica 'cero' (1ª versión)



C++
OBJETOS

- **Instanciación de objetos**
  - Una vez definida, una clase se puede instanciar, igual que cualquier tipo predefinido, para crear objetos:

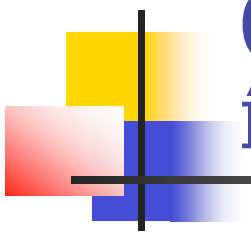
```
Fecha f1;
// Crea objeto f1 de tipo Fecha.
Fecha *f2;
f2 = new Fecha;
// f2: un objeto Fecha en memoria dinámica
```



C++
ACCESO

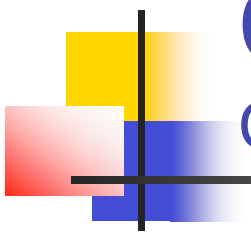
- **Acceso a propiedades de un objeto**
  - Igual que en un 'struct':

```
Fecha f1;
f1.dia = 10;
f1.setDia(10);

Fecha *f2;
f2 = new Fecha;
f2->dia = 10;
f2->setDia(10);
```



C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. **Estructura de un programa**
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. Compilador: g++
9. Directivas de compilación
10. Make
11. Práctica 'cero' (1ª versión)



C++
CLASES : DEFINICIÓN EN C++

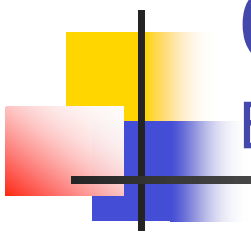
PARTES DE UN PROGRAMA:

- **Declaración de clases (fichero *.h*)**
  - **Datos miembro:** tipo y nombre del campo o atributo.
  - **Funciones/operadores miembro**: prototipos de los métodos de la clase.
- **Definición: implementación de métodos (fichero *.cc*)**
  - Las *funciones y operadores miembro* de una clase se definen anteponiendo a su nombre el nombre de la clase y el ***operador de resolución de ámbito (::)*** :

```
void Fecha::setDia(int d)
{ /*implementación de la función miembro*/ }
```

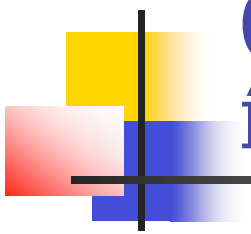
- **Definición del punto de entrada al programa (`main.cc`):**

```
int main(...) { ... }
```



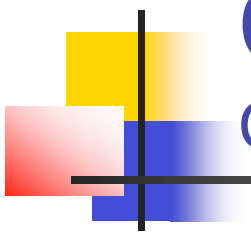
C++
ESTRUCTURA DE DIRECTORIOS

- **DISTRIBUCIÓN DEL CÓDIGO EN C++**
  - Declaraciones en .h: directorio **include**
  - Definiciones (implementación) de clases y código objeto: directorio **lib**
  - Punto de entrada (main.cc): directorio **src**
  - Documentación: directorio **doc**



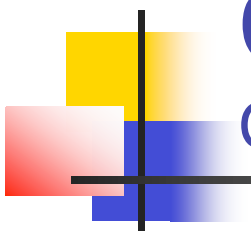
C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. **Operaciones *inline* y constantes**
7. Un ejemplo completo
8. Compilador: g++
9. Directivas de compilación
10. Make
11. Práctica 'cero' (1ª versión)



C++
CLASES : OPERACIONES *INLINE*

- Métodos ***inline***
  - Métodos a los que no se llama realmente, sino que el compilador inserta su código en el lugar de cada llamada.
- Declaración de métodos *inline*
  - Cuerpo directamente en la declaración de la clase (.h)
  - Limitar a métodos de una o dos líneas de código.

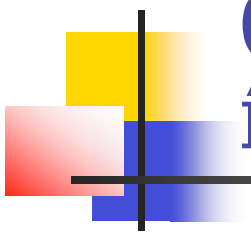


C++
CLASES : OPERACIONES *CONSTANTES*

- Métodos ***constantes***:
  - Funciones miembro que no modifican el contenido de un objeto (p. ej., métodos *get*)
- Se declaran con la palabra **`const`** tras la lista de argumentos:

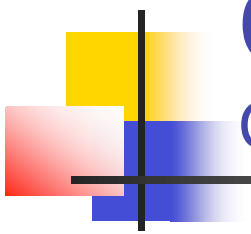
```
class Fecha {
 …
   int getDia() const;
}
…
int
Fecha::getDia() const {
   return dia;
}
```

```
const Fecha f;
…
f.getDia(); // OK, getDia() const
f.setDia(10); // ERROR, no const

Fecha f2;
f2.setDia(19); //OK
```



C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. **Un ejemplo completo**
8. Compilador: g++
9. Directivas de compilación
10. Make
11. Práctica 'cero' (1ª versión)



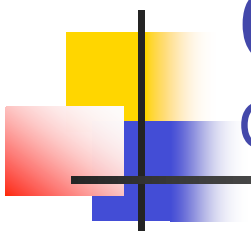
C++
CLASES : EJEMPLO

Clase 'rectangulo'

```
//rectangulo.h
class rectangulo {
public:
   void dimensiones(int,int);
   int area() const
   // método inline
   {
        return base*altura;
   }
private:
   int base, altura;
};
```

```
//rectangulo.cc
#include "rectangulo.h"
void rectangulo::dimensiones(int b,int h){
            base=b;
            altura=h;
}
```

```
//main.cc
#include <iostream>
#include "rectangulo.h"
using namespace std;

int main(){
            rectangulo r;     // declaro
   objeto
            r.dimensiones(3,5);// defino
   tamaño
            cout << "Area: " << r.area();
}
```



C++
CLASES : EJERCICIO

Ejercicio

1.- Ignorad la división en ficheros, y definid este código en un solo fichero llamado **todo.cc**

Tendréis que eliminar directiva #include "rectangulo.h"

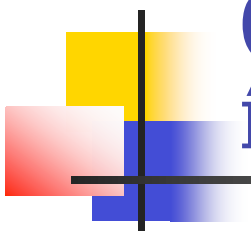
2.- Compilad el programa con **g++ –o rect todo.cc**

3.- Dividid ahora ese código en los tres ficheros especificados (rectangulo.h, rectangulo.cc y main.cc)

4.- Compilad el programa de nuevo con

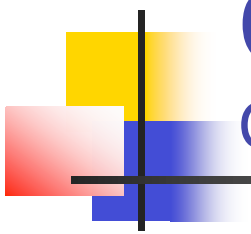
g++ –o rect2 rectangulo.cc main.cc

5.- Probad a hacer lo mismo poniendo cada fichero en su directorio correspondiente (.h en *include,* .cc en *lib* y main.cc en *src*)



C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. **Compilador: g++**
9. Directivas de compilación
10. Make
11. Práctica 'cero' (1ª versión)



C++

COMPILADOR: G++

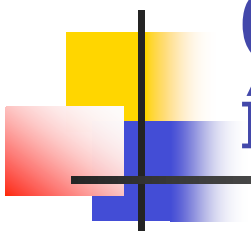
- GCC es una colección de compiladores para varios lenguajes: *C, C++, Objective C, Fortran y Java.* Incluye las librerías para éstos. Para C++ usaremos el compilador **g++.**

Sintaxis

```
$ g++ [opciones] nom_fichero...
```

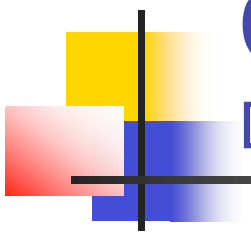
Opciones: con 'man gcc' podemos ver todas.

- **-help** → Indica a gcc que muestre su salida de ayuda
- **-o <fichero>** → El archivo ejecutable generado por gcc es por defecto a.out. Mediante este modificador, le especificamos el nombre del ejecutable.
- **-Wall** → No omite la detección de ningún warning. Por defecto, gcc omite una colección de warnings ``poco importantes''.
- **-c** → Preprocesa, compila y ensambla, pero no enlaza.
- **-g** → Incluye en el binario información necesaria para utilizar un depurador (**gdb** o **ddd**) posteriormente.



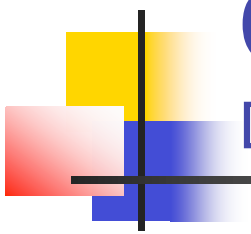
C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. Compilación: g++
9. **Directivas de compilación**
10. Make
11. Práctica 'cero' (1ª versión)



C++
DIRECTIVAS DE COMPILACIÓN

- Las directivas de compilación controlan el comportamiento del preprocesador, de forma que este puede ignorar o compilar determinadas líneas del código en función de ciertas condiciones que son evaluadas durante el preproceso.



C++
DIRECTIVAS DE COMPILACIÓN CONDICIONAL

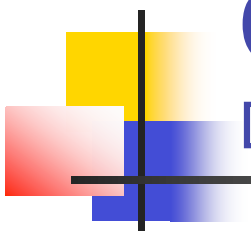
- **#ifdef…#endif, #ifndef…#endif**
  - Son condicionales especializadas en comprobar si un macro-identificador está definido (con **#define**) o no.

Sintaxis

```
#ifdef identificador … #endif
#ifndef identificador … #endif
```

Si la condición es cierta se compila lo que hay entre **#if…** y **#endif**. Si no se ignora.

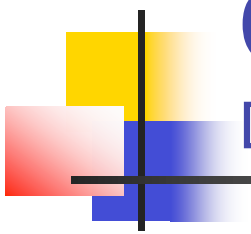
- Un macro-identificador X se define con **#define X** y se anula su definición con **#undef X**, con lo que podemos controlar a voluntad las zonas de código en que **X** se considera definido e indefinido.



C++
DIRECTIVAS DE COMPILACIÓN

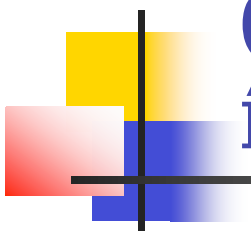
- La siguiente situación es muy habitual en proyectos en C++:
  - El fichero B.h incluye a A.h
  - El fichero C.h incluye a A.h
  - El fichero D.cc incluye a B.h y a C.h
- Para evitar problemas al compilar por la doble inclusión de ficheros de declaraciones (A.h en el ejemplo) se inserta las siguientes directivas en cada .h:

```
//Principio del fichero .h
#ifndef __A_H
#define __A_H
// … declaraciones en el .h …
#endif
// Fin del fichero .h
```



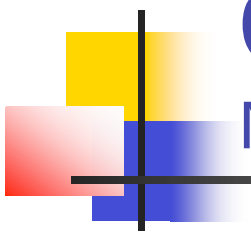
C++
DIRECTIVAS DE COMPILACIÓN

- **Ejercicio**:
  - Añade a *rectangulo.cc* un segundo
    `#include "rectangulo.h"`
  - Intenta compilar el código de nuevo. ¿Qué ocurre?
  - Modifica los ficheros *rectangulo.h, rectangulo.cc y main.cc* con las directivas `#ifdef, #ifndef, #endif` que consideres necesarias. Compila y comprueba que todo funciona correctamente.



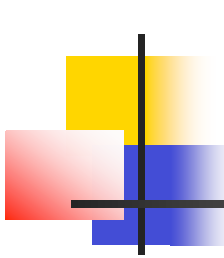
C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. Compilador: g++
9. Directivas de compilación
10. **Make**
11. Práctica 'cero' (1ª versión)



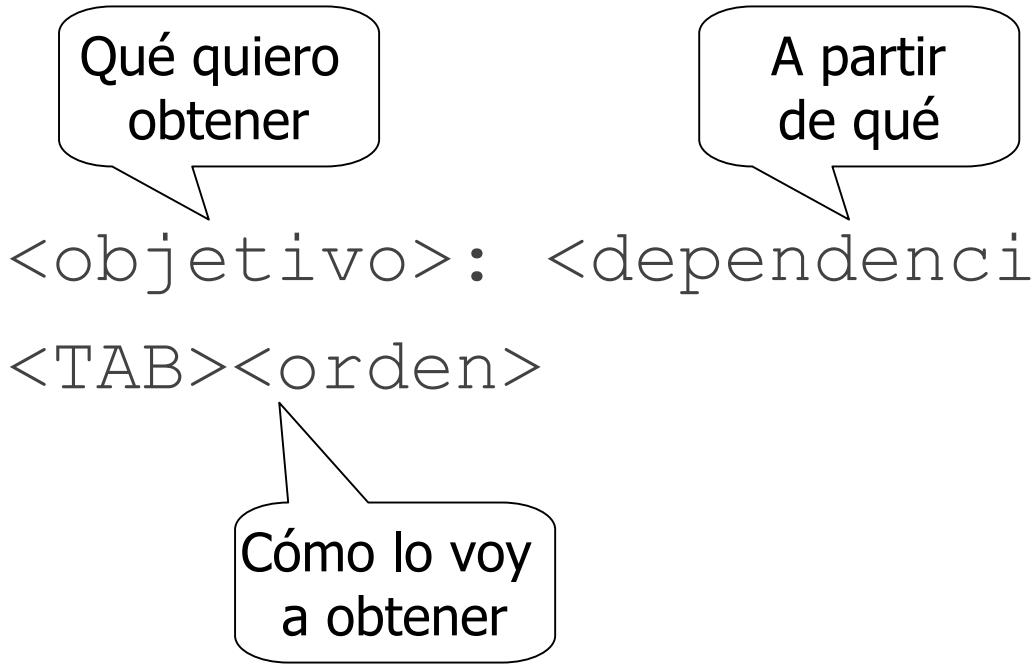
C++
MAKE

- `make`: Utilidad que automatiza el proceso de compilación
- Busca un fichero de configuración llamado `makefile` o `Makefile` (en ese orden)
- Ejemplos:
  - `make`
  - `make prog1 #crea el objetivo prog1`
  - `make -f mimakefile` (fich. config. mimakefile)
  - `make -f mimakefile prog1`



C++
MAKE

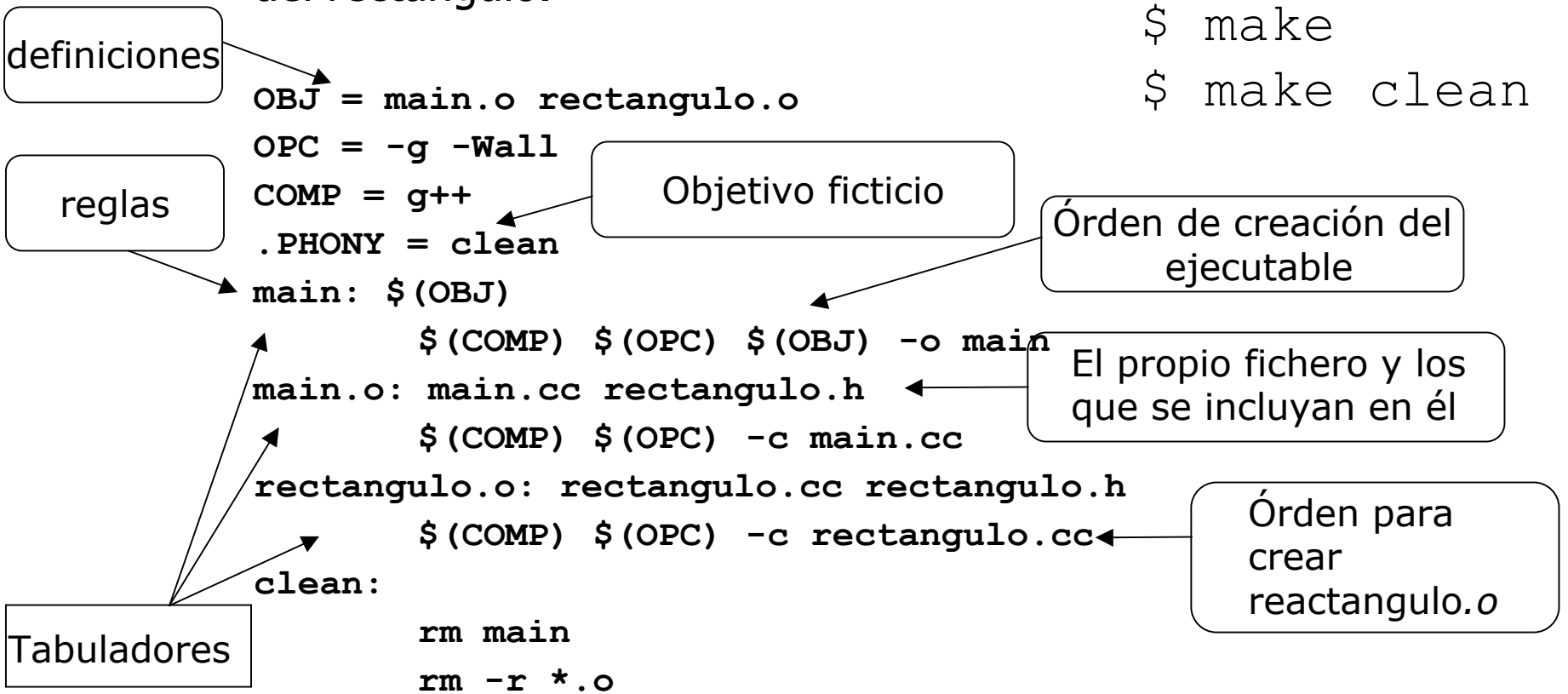
- Un fichero 'makefile' contiene reglas con este formato:

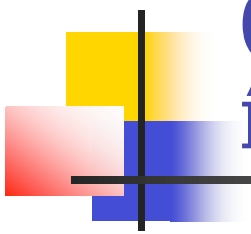


C++
El fichero de configuración `Makefile`

- Makefile de ejemplo que automatiza la compilación del ejemplo del rectángulo.

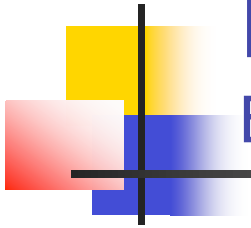
```
$ make
$ make clean
```





C++
ÍNDICE

1. Clases en C++: estructura, ámbito de variables.
2. Declaración y definición de clases
3. Instanciación de objetos
4. Acceso a propiedades de un objeto
5. Estructura de un programa
6. Operaciones *inline* y constantes
7. Un ejemplo completo
8. Compilador: g++
9. Directivas de compilación
10. Make
11. **Práctica 'cero' (1ª versión)**

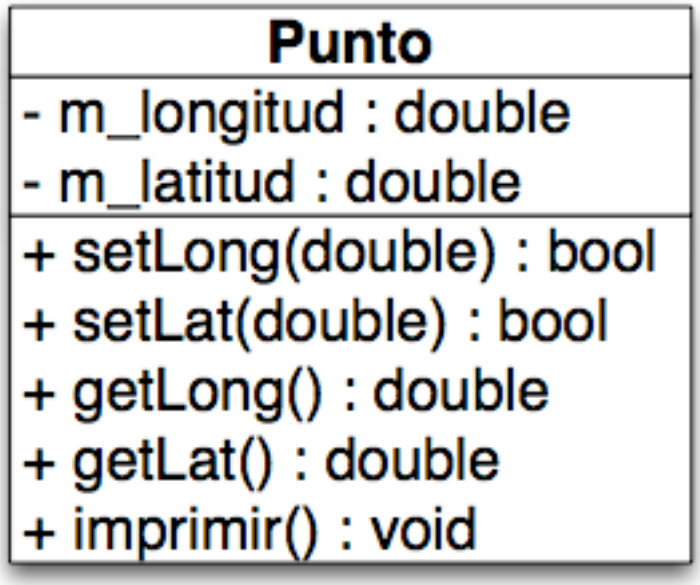


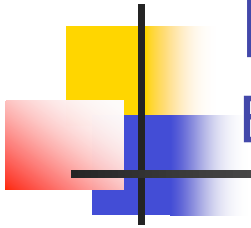
PRÁCTICA 0
ENUNCIADO

Los objetivos de esta pequeña práctica son:

- Introducirnos en la P.O.O.
- Implementar una clase sencilla
- Familiarizarnos con la estructura de directorios del código fuente y el proceso de compilación.
- Realizar una entrega de prueba en el servidor de prácticas del DLSI
- Su entrega es obligatoria pero no se evaluará.

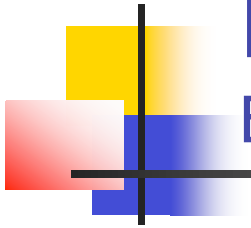
PRÁCTICA 0
ENUNCIADO





PRÁCTICA 0
ENUNCIADO

- Dado el diagrama de clase de la figura anterior:
  - Implementa la clase **Punto** mediante dos ficheros
    - *Punto.h*: declaración de métodos y variables. El fichero debe ubicarse en el directorio **include**
    - *Punto.cc*: implementación de métodos. Este fichero debe ubicarse en el directorio **lib**
  - Identifica los métodos constantes y declaralos como tales.



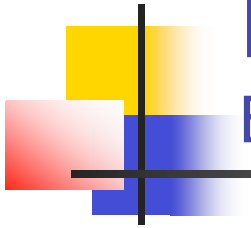
PRÁCTICA 0
ENUNCIADO

- Crea un fichero *main.cc* con una función `main()` donde se pidan al usuario un par de números con los que debéis crear un punto que debéis imprimir por pantalla. Escribe el código necesario para que cada método público de la clase sea utilizado al menos una vez. Este fichero debe ubicarse en el directorio **src**.

- El método P`unto::imprimir()` debe imprimir los valores 'm_longitud' y 'm_latitud' separados por una coma por la salida estándar.

- Comprimid toda la estructura de directorios en un fichero llamado **p0-10-11.tgz**

  - `tar -czvf p0-10-11.tgz *`



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.

PRÁCTICA 0
FICHERO MAKEFILE

```
# Este fichero no se debe modificar. Lo puedes encontrar en la sección de materiales del CV.
# La practica debe compilar con este fichero situado en el directorio raíz del proyecto, ejecutando
# la orden 'make'.
#
COMP=g++
OPC=-g -Wall

.PHONY=clean

all: main

main: ./src/main.cc Punto.o ./include/Punto.h
    $(COMP) $(OPC) ./src/main.cc Punto.o -I ./include -o main

Punto.o: ./lib/Punto.cc ./include/Punto.h
    $(COMP) $(OPC) -c ./lib/Punto.cc -I ./include

clean:
    rm main
    rm -r *.o
```