

CLASES Y OBJETOS

Clases y Objetos. Introducción (I).

- Programación estructurada ->
Procedimientos y Datos separados
DATOS → PROCEDIMIENTOS → RESULTADOS
- Programación Orientada a Objetos -> Datos
y Funciones encapsulados en estructuras
llamadas **Clases** → Modelan objetos del
mundo real.
 - Instanciación → **Objetos**

Clases y Objetos. Introducción (II)

- Clase TCoordenada

- Datos: x,y,z (atributos)
- Metodos: (comportamientos)
 - LeerX
 - LeerY
 - LeerZ
 - PonerX
 - PonerY
 - PonerZ

Clases y Objetos. Introducción (III)

● Instanciaciones de la clase TCoordenada

```
1 // Declaración de un objeto  
2 TCoordenada objetoP1;  
3 // Declaración de un array  
4 TCoordenada arrayP2[10];  
5 // Declaración de un puntero  
6 TCoordenada *ptrP3 = &objetoP1;  
7 // Declaración de una referencia  
8 TCoordenada &refP4 = objetoP1;
```

Clases y Objetos. Introducción (IV)

- En un programa principal:

```
1 #include <iostream>           Ejemplo 2.1 _____  
2  
3 using namespace std;         Fichero encabezado  
4  
5 int  
6 main(void)  
7 {  
8     int i;  
9     TCoordenada p1;          Espacio de nombres  
10  
11    return 0;  
12 }
```

Instanciación de la clase TCoordenada

Clases y Objetos. Introducción (V)

- Compilación (Preprocesador, Compilador, Enlazador)
- Resultado compilacion ejemplo 2.1:

Salida ejemplo 2.1

```
1 ejem1.cc: In function ‘int main()’:  
2 ejem1.cc:9: ‘TCoordenada’ undeclared (first use this function)  
3 ejem1.cc:9: (Each undeclared identifier is reported only once for  
4     each function it appears in.)  
5 ejem1.cc:9: syntax error before ‘;’ token
```

Clases y Objetos.

Declaración de una clase (I)

```
1 | class NombreClase {  
2 | // Contenido de la clase  
3 | };
```

```
1 | class TCoordenada {  
2 |     int x, y, z;  
3 | };
```

Clases y Objetos.

Declaración de una clase (II)

- Añadiendo al main la declaración:

Ejemplo 2.2

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     int x, y, z;
7 };
8
9 int
10 main(void)
11 {
12     int i;
13     TCoordenada p1;
14
15     return 0;
16 }
```

Clases y Objetos.

Acceso a los miembros de una clase (I)

- Mediante “.” o “->”

```
1   TCoordenada p1;
2
3   cout << "Componente x: " << p1.x << endl;
4   cout << "Componente y: " << p1.y << endl;
5   cout << "Componente z: " << p1.z << endl;
```

```
1   TCoordenada p1;
2   TCoordenada *ptr1 = &p1;
3
4   cout << "Componente x: " << ptr1->x << endl;
5   cout << "Componente y: " << ptr1->y << endl;
6   cout << "Componente z: " << ptr1->z << endl;
```

Clases y Objetos.

Acceso a los miembros de una clase (II)

- En el programa principal podemos añadir:

Ejemplo 2.3

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     int x, y, z;
7 };
8
9 int
10 main(void)
11 {
12     int i;
13     TCoordenada p1;
14
15     p1.x = 1;
16     p1.y = 2;
17     p1.z = 3;
18
19     return 0;
20 }
```

Clases y Objetos.

Acceso a los miembros de una clase (III)

- Pero al compilar...

Salida ejemplo 2.3 ■

```
1 ejem3.cc: In function ‘int main()’:  
2 ejem3.cc:6: ‘int TCoordenada::x’ is private  
3 ejem3.cc:15: within this context  
4 ejem3.cc:6: ‘int TCoordenada::y’ is private  
5 ejem3.cc:16: within this context  
6 ejem3.cc:6: ‘int TCoordenada::z’ is private  
7 ejem3.cc:17: within this context
```

Clases y Objetos. Control de Acceso (I)

- Ocultación de Información.
- ¿Para qué? → Independencia de la implementación.
- Ejemplo: Distintas implementaciones de la clase hora

```
1 // nSegundos = 920
2 class THora {
3     int nSegundos;
4 }
5
6 // hora = "00:15:20"
```

Clases y Objetos. Control de Acceso (II)

```
7   class THora {  
8     char *hora;  
9   };  
10  // beat = 7.1 ← Errata. Es 10.6 beats  
11  class THora {  
12    float beat;  
13  };  
14  
15  // horas = 0, minutos = 15, segundos = 20  
16  class THora {  
17    int horas, minutos, segundos;  
18  };  
19
```

Clases y Objetos. Control de Acceso (III)

- Accesos:

	Descripción
public:	Accesible tanto desde la propia clase como desde funciones ajenas a la clase
private:	Accesible exclusivamente desde las funciones miembros y funciones y clases amigas
protected:	Se emplea para limitar el acceso a las clases derivadas, su funcionamiento depende del tipo de herencia que se realice

Clases y Objetos.

Control de Acceso (IV)

```
1  class UnaClase {  
2      public:  
3          // Parte pública  
4          // Normalmente, sólo funciones  
5  
6      protected:  
7          // Parte protegida  
8          // Funciones y datos  
9  
10     private:  
11         // Parte privada  
12         // Normalmente, datos y funciones auxiliares
```

Clases y Objetos.

Control de Acceso (V)

- Modificando el ejemplo 2.3

Ejemplo 2.4

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     public:
7         int x, y, z;
8 }
9
10 int
11 main(void)
12 {
13     int i;
14     TCoordenada p1;
15
16     p1.x = 1;
17     p1.y = 2;
18     p1.z = 3;
19
20     cout << "(" << p1.x << ", " << p1.y << ", " << p1.z << ")" << endl;
21
22     return 0;
23 }
```

Los hacemos públicos para poder acceder desde el main

(1, 2, 3)

Clases y Objetos. Control de Acceso (VI)

- Inconveniente: Acceso público a la implementación.
- Solución:
 - Hacer privados los datos
 - Permitir el acceso a ellos a través de métodos públicos

Clases y Objetos.

Control de Acceso (VII)

Ejemplo 2.5

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     public:
7         void setX(int xx) {x = xx;}
8         void setY(int yy) {y = yy;}
9         void setZ(int zz) {z = zz;}
10
11        int getX(void) {return x;}
12        int getY(void) {return y;}
13        int getZ(void) {return z;}
14
15    private:
```

Funciones inline

```
16             int x, y, z;
17         };
18
19         int
20         main(void)
21         {
22             int i;
23             TCoordenada p1;
24
25             p1.setX(1);
26             p1.setY(2);
27             p1.setZ(3);
28
29             cout << "(" << p1.getX();
30             cout << ", " << p1.getY();
31             cout << ", " << p1.getZ();
32             cout << ")" << endl;
33
34             return 0;
35 }
```

(1, 2, 3)

Clases y Objetos.

Visualización de un objeto (I)

- Para ver un objeto TCoordenada:

```
1 | cout << "(" << p1.getX();
2 | cout << ", " << p1.getY();
3 | cout << ", " << p1.getZ();
4 | cout << ")" << endl;
```

- Por sobrecarga del operador ‘cout’ (se verá más adelante)
- Creando un método para la clase:

Clases y Objetos.

Visualización de un objeto (II)

Ejemplo 2.6

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     public:
7         void setX(int xx) {x = xx;}
8         void setY(int yy) {y = yy;}
9         void setZ(int zz) {z = zz;}
10
11     int getX(void) {return x;}
12     int getY(void) {return y;}
13     int getZ(void) {return z;}
14
15     void Imprimir(void) {
16         cout << "(" << x;
17         cout << ", " << y;
18         cout << ", " << z;
19         cout << ")";
20     }
21
22     private:
23         int x, y, z;
24 }
```

```
25
26     int
27 main(void)
28 {
29     int i;
30     TCoordenada p1, p2;
31
32     p1.setX(1);
33     p1.setY(2);
34     p1.setZ(3);
35
36     p2.setX(4);
37     p2.setY(5);
38     p2.setZ(6);
39
40     p1.Imprimir();
41     cout << endl;
42     p2.Imprimir();
43     cout << endl;
44
45     return 0;
46 }
```

(1, 2, 3)
(4, 5, 6)

Clases y Objetos. Empleo de punteros

Ejemplo 2.7 —

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     public:
7         int x, y, z;
8 };
9
10 int
11 main(void)
12 {
13     int i;
14     TCoordenada p1;
15     TCoordenada *ptr1 = &p1;
16
17     ptr1->x = 1;
18     *ptr1.x = 1; ← Solución: (*ptr1).x = 1
19
20     return 0;
21 }
```

Salida ejemplo 2.7 —

```
1 ejem6.cc: In function ‘int main()’:
2 ejem6.cc:18: request for member ‘x’ in ‘ptr1’, which is of
3     non-aggregate type ‘TCoordenada*’
```

Clases y Objetos.

Separación de la interfaz y la implementación (I)

- Facilita el mantenimiento de la clase
- ¿Cómo separarlo?
 - La declaración de la clase (interfaz) en un archivo de cabecera (.h)
 - La implementación de la clase en un archivo fuente (.cc o .cpp)
- Al usuario se le proporciona:
 - Archivo cabecera → uso con #include
 - Archivo objeto del archivo fuente (.o) → Incluir en el linkado

Clases y Objetos.

Separación de la interfaz y la implementación (II)

- En tcoordenada.h:

Ejemplo 2.8

```
1 #include <iostream>
2
3 using namespace std;
4
5 class TCoordenada {
6     public:
7         void setX(int);
8         void setY(int);
9         void setZ(int);
10
11         int getX(void);
12         int getY(void);
13         int getZ(void);
14
15         void Imprimir(void);
16
17     private:
18         int x, y, z;
19 }
```

← Ya no son funciones inline

Clases y Objetos.

Separación de la interfaz y la implementación (III)

- Implementación en tcoordenada.cc

Ejemplo 2.9 -

```
1 #include "tcoordenada.h"
2
3 void
4 TCoordenada::setX(int xx) {
5     x = xx;
6 }
7
8 void
9 TCoordenada::setY(int yy) {
10    y = yy;
11 }
12
13 void
14 TCoordenada::setZ(int zz) {
15    z = zz;
16 }
17
```

```
18 int
19 TCoordenada::getX(void) {
20     return x;
21 }
22
23 int
24 TCoordenada::getY(void) {
25     return y;
26 }
27
28 int
29 TCoordenada::getZ(void) {
30     return z;
31 }
32
33 void
34 TCoordenada::Imprimir(void) {
35     cout << "(" << x;
36     cout << ", " << y;
37     cout << ", " << z;
38     cout << ")";
39 }
```

Clases y Objetos.

Separación de la interfaz y la implementación (IV)

- Observar operador ámbito :: en funciones de la implementación
- Probar la clase en un programa principal

Ejemplo 2.10 –

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int
8 main(void)
9 {
10     int i;
11     TCoordenada p1;
12
13     p1.setX(1);
14     p1.setY(2);
15     p1.setZ(3);
16
17     p1.Imprimir();
18     cout << endl;
19
20     return 0;
21 }
```

La salida es:
→ (1, 2, 3)

Clases y Objetos.

Separación de la interfaz y la implementación (IV)

- Compilación y enlazado:

```
1 | g++ -c tcoordenada.cc           ← Compilación
2 | g++ -c main.cc                  ← Enlazado
3 | g++ -o main main.o tcoordenada.o
```

Clases y Objetos.

La herramienta make (I)

- Intérprete de ficheros de texto de propósito general.
- Nuestro uso: para compilar y enlazar nuestros programas
- El fichero de texto a interpretar ha de seguir una estructura determinada:

*Objetivo: Dependencia/s
(tabulador) Orden a ejecutar*

Clases y Objetos. La herramienta make (II)

- Si dependencias no resueltas, se convierten en subobjetivos:

Objetivo: Dep1 Dep2 ...

(tabulador) Orden a ejecutar

Dep1: Dep11 Dep12 ...

(tabulador) Orden1 a ejecutar

Dep2: Dep21 Dep22 ...

(tabulador) Orden2 a ejecutar

...

Clases y Objetos.

La herramienta make (III)

- Aplicado a compilación de ficheros en C++:
ejecutable: *fich1.o fich2.o ...*
 (→) *g++ -o ejecutable fich1.o fich2.o ...*
fich1.o: *fich11.cc fich11.h ...*
 (→) *g++ -c -o fich1.o fich11.cc ...*
fich2.o: *fich21.cc fich21.h ...*
 (→) *g++ -c -o fich2.o fich21.cc ...*
 ...

Clases y Objetos.

La herramienta make (IV)

- Ejemplo:

- Fichero cabecera: ***tcoordenada.h***
- Ficheros implementación: ***tcoordenada.cc***
main.cc

Ejemplo 2.11 —

```
1 main: main.o tcoordenada.o
2           g++ -o main main.o tcoordenada.o
3
4 main.o: main.cc tcoordenada.h
5           g++ -c main.cc
6
7 tcoordenada.o: tcoordenada.h tcoordenada.cc
8           g++ -c tcoordenada.cc
```

Clases y Objetos.

La herramienta make (V)

- Usando variables:

Ejemplo 2.12 —

```
1 OBJ=main.o tcoordenada.o
2 COMP=g++
3 OPC=-g
4
5 main: $(OBJ)
       $(COMP) $(OPC) $(OBJ) -o main
7
8 main.o: main.cc tcoordenada.h
9      $(COMP) $(OPC) -c main.cc
10
11 tcoordenada.o: tcoordenada.h tcoordenada.cc
12      $(COMP) $(OPC) -c tcoordenada.cc
```

Clases y Objetos.

Ficheros de encabezado.

- De C : Llevan prefijo c -> cstdio, cstdlib, cstring,...
- No llevan extensión .h
- Los estándar de C y C++ en el include van entre <>:
`#include <iostream>`
`#include <cstring>`
`#include <string>`

Clases y Objetos.

Uso de espacios de nombres (I)

- En proyectos de gran tamaño con uso de librerías externas → Problemas de colisión de nombres de funciones, variables globales, clases, etc.
- En C++ solución: Uso ***espacio de nombres***
- Declaración:

```
namespace Espacio {  
    <declaración de variables, clases, funciones, etc>  
}
```

Clases y Objetos.

Uso de espacios de nombres (II)

- Acceso al elemento.Tres formas:
 - espacioNombre::miembro
 - using espacioNombre::miembro
 - using namespace espacioNombre
- Ejemplo:

```
namespace Espacio1 {  
    int a;  
}  
  
namespace Espacio2 {  
    int a;  
}
```

```
using namespace Espacio2;  
Espacio1::a=3;  
a=5; //Es Espacio2::a
```

- Librerías de C++ dentro del espacio de nombres **Std**

using namespace std;