

Metodología de la Programación C++

Tema 3.- Clases y abstracción

Abstracción y encapsulamiento

Introducción a las clases en C++

- Atributos y métodos**
- Diferencias con struct**
- Métodos const**
- El puntero this**
- Funciones y clases amigas**
- Partes públicas y privadas**
- El operador .**
- Operador :: (resolución de ámbito)**
- Métodos inline**

ugr Universidad de Granada

DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 1

Abstracción y encapsulamiento C++

¿Qué es la abstracción?

Conseguir que los programas se abstraigan u obvien el funcionamiento interno de lo que usan.

Abstraer (según DRAE):

1. Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.
2. Prescindir, hacer caso omiso.

Ejemplo: abstracción procedural

```
main() {
    int x, y;
    cin >> x;
    y = factorial(x);
    cout << "El factorial de " << x << " vale " << y;
}
```

Nos da igual como funcione internamente la función **factorial()**
main() es más fácil de construir/leer/mantener al no tener que conocer los detalles de **factorial()**

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 2



Abstracción y encapsulamiento

¿Qué es la abstracción?

C++

Ejemplo: abstracción procedural

```
main() {  
    int x, y;  
    cin >> x;  
    y = factorial(x);  
    cout << "El factorial de " << x << " vale " << y;  
}
```

```
int factorial(int n) {  
    int f=1;  
    for (int i=2; i<=n; i++)  
        f = f*i;  
    return f;  
}
```

```
int factorial(int n) {  
    int f=1, i=2;  
    while (i<=n)  
        f *= i++;  
    return f;  
}
```

La implementación y el comportamiento de `main()` es el mismo:
Se **abstira** de la implementación de `factorial()`
Sólo necesita conocer la **interfaz (prototipo)** de `factorial()`

Abstracción y encapsulamiento

¿Hay más tipos de abstracción?

¿Podemos abstraernos de la representación de los datos?

¿Podemos hacer que un programa sea independiente de la forma en la que se representan los datos en la memoria?

Construir un **struct** que permita almacenar fechas:

```
struct Fecha {  
    int d, m, a;  
};
```

Implementar una función que calcule la diferencia entre dos fechas:

- El cálculo se hace en días.
- Suponemos que todos los meses tienen 30 días.
- Suponemos que no hay años bisiestos.

```
int diferencia(const Fecha f1, const Fecha f2);
```

Abstracción y encapsulamiento C++
La abstracción de datos

Implementamos la función `diferencia()`

```

int diferencia(const Fecha f1, const Fecha f2)
{
    int difd=0, difm=0, difa=0;

    difd = f2.d-f1.d;
    if (difd<0) {
        difd += 30;
        difm -= 1;
    }
    difm += f2.m-f1.m;
    if (difm<0) {
        difm += 12;
        difa -= 1;
    }
    difa += f2.a-f1.a;
    return (difa<0) ? (-1) : (difd+difm*30+difa*360);
}

```

struct Fecha {
 int d, m ,a;
};

{ // Otra implementación
 int d1, d2;
 d1 = f1.d + 30*f1.m + 30*12*f1.a;
 d2 = f2.d + 30*f2.m + 30*12*f2.a;
 return (d2<d1) ? (-1) : (d2-d1);
}

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 5

Abstracción y encapsulamiento C++
La abstracción de datos

```

int main(int argc, char *argv[])
{
    Fecha f1, f2;
    int dif;

    cout << "Dime una fecha: ";
    cin >> f1.d >> f1.m >> f1.a;
    NO hay abstracción de datos

    cout << "Dime una fecha posterior: ";
    cin >> f2.d >> f2.m >> f2.a;
    Abstracción procedural

    if ((dif=diferencia(f1,f2))>=0)
        cout << "Hay " << dif << ".dias de diferencia" << endl;
    else
        cout << "La segunda fecha debe ser posterior" << endl;
}

```

¿Y si cambiamos el tipo `Fecha`? → Conceptualmente, seguiremos haciendo lo mismo, pero tenemos que cambiar cosas ...

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Universidad de Granada) [Navigation icons] 6

Abstracción y encapsulamiento C++

La abstracción de datos

Cambiamos la representación de **Fecha**:
(Número de días transcurridos desde 1/1/1900)

```

struct Fecha {
    int dias;
};

int diferencia(const Fecha f1, const Fecha f2) {
    return (f2.dias<f1.dias) ? -1 : (f2.dias-f1.dias);
}

int main(int argc, char *argv[]) {
    Fecha f1, f2;      int dif, d, m, a;
    cout << "Dime una fecha: ";
    cin >> d >> m >> a;
    f1.dias = d+30*m+(a-1900)*360;
    cout << "Dime una fecha posterior: ";
    cin >> d >> m >> a;
    f2.dias = d+30*m+(a-1900)*360;
    if ((dif=diferencia(f1,f2))>=0)
        cout << "Hay " << dif << " días de diferencia" << endl;
    else
        cout << "La segunda fecha debe ser posterior" << endl;
}

```

Adaptamos
diferencia()

Se mantienen sin cambios
aquellos lugares donde
hemos usado abstracción

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 7

Abstracción y encapsulamiento C++

La abstracción de datos

```

int diferencia(const Fecha f1, const Fecha f2) {
    return (f2.dias<f1.dias) ? -1 : (f2.dias-f1.dias);
}

void LeeFecha(Fecha &f) {
    int d, m, a;
    cout << "Dime una fecha: ";
    cin >> d >> m >> a;
    f.dias = d+30*m+(a-1900)*360;
}

int main(int argc, char *argv[]) {
    Fecha f1, f2;
    int dif;

    LeeFecha(f1);
    LeeFecha(f2);

    if ((dif=diferencia(f1,f2))>=0)
        cout << "Hay " << dif << " días de diferencia" << endl;
    else
        cout << "La segunda fecha debe ser posterior" << endl;
}

```

struct Fecha {
 int dias;
};

main() ya no depende de la
representación concreta de
Fecha

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 8

Abstracción y encapsulamiento C++

Los Tipos de Datos Abstractos



Para que cualquier programa se **abstraiga**, por completo, de la representación de un tipo de dato construimos un **T.D.A.**

Representación de datos
 +
 Operaciones

```

struct Fecha {
    int dias; // Días transcurridos desde 1/1/1900
};

void GetFecha(const Fecha f, int &d, int &m, int &a)
{
    a = f.dias / 360;
    m = (f.dias%360) / 30;
    d = (f.dias%360) % 30;
}

void SetFecha(Fecha &f, const int d, const int m, const int a)
{
    f.dias = d + m*30 + a*360;
}
  
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 9

Abstracción y encapsulamiento C++

Los Tipos de Datos Abstractos



Hacemos uso del TDA Fecha:

```

void LeeFecha(Fecha &f)
{
    int d, m, a;
    cout << "Dime la fecha: ";
    cin >> d >> m >> a;
    SetFecha(f,d,m,a);
}

int diferencia(const Fecha f1, const Fecha f2)
{
    int f1d, f1m, f1a, f2d, f2m, f2a, d1, d2;
    GetFecha(f1, f1d, f1m, f1a);
    GetFecha(f2, f2d, f2m, f2a);
    d1 = f1d + 30*f1m + 30*12*f1a;
    d2 = f2d + 30*f2m + 30*12*f2a;
    return (d2<d1) ? (-1) : (d2-d1);
}
  
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 10



Tema 5.- Clases y abstracción

Abstracción y encapsulamiento

Introducción a las clases en C++

Atributos y métodos	Partes públicas y privadas
Diferencias con struct	El operador .
Métodos const	Operador :: (resolución de ámbito)
El puntero this	Métodos inline

Funciones y clases amigas

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 12

Introducción a las clases en C++ C++

Advertencia

Para explicar la construcción de las clases en C++ se ha usado un **enfoque constructivo** en lugar de usar un enfoque enumerativo.

Desarrollaremos la sintaxis y conceptos de las clases siguiendo un razonamiento lógico sobre la **necesidad** de dichas ideas en base a lo que ya se conoce.

En las transparencias sucesivas (este tema y siguientes relacionados con clases) hay **múltiples errores sintácticos (intencionados)** que irán dando pie a nuevas formas de sintaxis de forma sucesiva.

«*Todo lo que nace proviene necesariamente de una causa; pues sin causa nada puede tener origen*»

Platón

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | 13

Introducción a las clases en C++ C++

Las clases

Disponer del TDA no impide, **sintácticamente**, saltarse el interfaz para acceder a la representación interna.

```

Representación interna
Interfaz
Rompe la abstracción
TDA
Aplicación

```

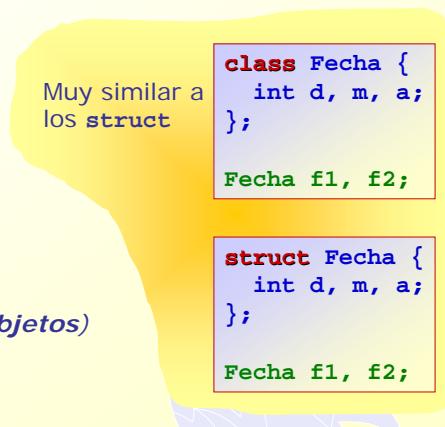
```

Representación interna:
Interfaz:
Rompe la abstracción:
TDA:
Aplicación:

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | 14

Introducción a las clases en C++ 
Las clases



Las clases nos van a ayudar a conseguir nuestro objetivo: **mantener el encapsulamiento y evitar romper la abstracción.**

Definición

```
class <NOMBRE> {
    <atributos>
    <métodos>
};
```

Declaración de variables (objetos)

```
<NOMBRE> variables;
```

```
class Fecha {
    int d, m, a;
};

Fecha f1, f2;
```

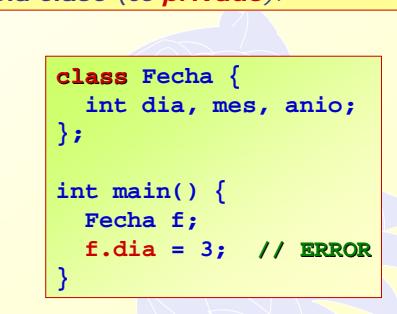
```
struct Fecha {
    int d, m, a;
};

Fecha f1, f2;
```

Aunque las clases son mucho más, en este curso nos limitaremos a verlas simplemente como una **herramienta de programación**.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  15

Introducción a las clases en C++ 
Diferencia entre class y struct



Por defecto, lo que hay dentro de un **struct** puede ser accedido desde cualquier parte (es **público**).

*Por defecto, lo que hay dentro de una clase (**class**) sólo puede ser accedido desde dentro de la propia clase (es **privado**).*

```
struct Fecha {
    int dia, mes, anio;
};

int main() {
    Fecha f;
    f.dia = 3; // OK
}
```

```
class Fecha {
    int dia, mes, anio;
};

int main() {
    Fecha f;
    f.dia = 3; // ERROR
}
```

No hay **ninguna otra diferencia**. En este curso procuraremos usar **class** (propio de C++) en lugar de **struct** (se asocia más con C).

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  16

Introducción a las clases en C++ C++

Los métodos de la clase



Deseamos implementar las funciones `SetFecha` y `GetFecha`

```
void SetFecha(Fecha &f, const int d, const int m, const int a)
{
    f.dia = d;
    f.mes = m;
    f.anio = a;
}
```

Error: estamos tocando la parte privada !!!

```
class Fecha {
    int dia, mes, anio;
};
```

Solución: Meto las funciones dentro de la clase (**métodos**). Al estar dentro, ya sí pueden consultar o modificar los atributos de los objetos de la clase.

```
class Fecha {
    int dia, mes, anio;
    void SetFecha(Fecha &f, const int d, const int m, const int a);
    void GetFecha(const Fecha f, int &d, int &m, int &a);
};
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 17

Introducción a las clases en C++ C++

Parte pública de una clase



¿Es posible acceder a alguna parte de la clase **desde fuera** de la clase?

Ejemplo:
Construir una función que lea una fecha desde teclado.

```
void LeeFecha(Fecha &f) {
    int d, m, a;
    cout << "Dime la fecha: ";
    cin >> d >> m >> a;
    SetFecha(f,d,m,a);
```

*!!! SetFecha()
es privado !!!*

Podemos indicar que algunos componentes de la clase son públicos:

```
class Fecha {
    int dia, mes, anio; // Por defecto, privados
public: // A partir de aquí: públicos
    void SetFecha(Fecha &f, int d, int m, int a);
    void GetFecha(Fecha f, int &d, int &m, int &a);
}; // Se omiten los const por falta de espacio
```

*Los componentes privados se pueden indicar con **private**.*

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 18

Introducción a las clases en C++ 

Partes públicas y privadas

Podemos tener tantas partes **public** y **private** como queramos, aunque lo normal es agrupar por separado ambas partes.

```

class Fecha {
    // Por defecto, comenzamos con una sección private
    int dia;

    public:
        void SetFecha(Fecha &f, int d, int m, int a);

    private:
        int mes, anio;

    public:
        void GetFecha(Fecha f, int &d, int &m, int &a);
}; // Se omiten los const por falta de espacio

```

Suele ocurrir que:
La representación interna es **private.**
La interfaz es **public.**

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  19

Introducción a las clases en C++ 

El operador punto

¿Cómo accedemos a los miembros de una clase?
 Igual que con los **struct** (operador punto)
objeto.miembro

```

class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(Fecha &f, int d, int m, int a);
    void GetFecha(Fecha f, int &d, int &m, int &a);
}; // Se omiten los const por falta de espacio

```

```

...
Fecha f;
...
f.anio = 1954;
...
f.SetFecha(f,6,6,1944);
...

```

Tanto los **métodos** como los **atributos** están **dentro** de la clase

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  20

Introducción a las clases en C++ C++

El operador punto

```

class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(Fecha &f, int d, int m, int a);
    void GetFecha(Fecha f, int &d, int &m, int &a);
}; // Se omiten los const por falta de espacio

```

```

...
Fecha f;
...
f.anio = 1954;
...
f.SetFecha(f,6,6,1944);
...

```

.anio está referido a f
.SetFecha(...) está referido a f

Le estamos dando dos veces el mismo objeto: es **redundante**
Como **f.** hay que ponerlo siempre, podemos quitar el parámetro **f**

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 21

Introducción a las clases en C++ C++

Acceso a los miembros desde la clase

```

void SetFecha(Fecha &f, const int d, const int m, const int a) {
    f.dia = d;
    f.mes = m;
    f.anio = a;
}

```

Versión struct

```

struct Fecha {
    int dia, mes, anio;
};

```

¿Cómo nos referimos a los atributos si quitamos el parámetro?

```

void SetFecha(Fecha &f, const int d, const int m, const int a)
{
    f.dia = d;
    f.mes = m;
    f.anio = a;
}

```

Versión struct

```

class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(...);
};
...
Fecha f;
...
f.SetFecha(f,6,6,1944);

```

Versión class

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 22

Introducción a las clases en C++ C++

Métodos const

Ejemplo: Construir el método `GetFecha()`

```

void GetFecha(const Fecha f, int &d, int &m, int &a) {
    d = f.dia;
    m = f.mes;
    a = f.anio;
}

void GetFecha(int &d, int &m, int &a) {
    d = dia;
    m = mes;
    a = anio;
}

void GetFecha(int &d, int &m, int &a) const {
    d = dia;
    m = mes;
    a = anio;
}

```

Versión struct Versión class Versión class

Al pasar a esta nueva forma de escritura ...
¿Cómo podemos indicar el `const` si no hemos escrito el parámetro?

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 23

Introducción a las clases en C++ C++

El operador :: (resolución de ámbito)

```

class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(int d, int m, int a);
}; // Se omiten los const por falta de espacio

void SetFecha(int x, int y, int z); // Otra función

Fecha f;

f.SetFecha(6,6,1944); // La de la clase
SetFecha(1,4,1978); // La otra

```

La implementación `SetFecha()` es:

```

void SetFecha(int d, int m, int a)
{
    dia = d;
    mes = m;
    anio = a;
}

```

¿Cómo sabemos que esta es la implementación de la función de la clase y no de la otra?

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 24

Introducción a las clases en C++ C++

El operador :: (resolución de ámbito)

Para referirnos a cualquier componente de una clase podemos usar el **operador de resolución de ámbito ::**

Su uso es obligatorio cuando pudiese haber ambigüedad.

La forma de usarlo es:

```
NombreDeLaClase::Componente
```

```
void Fecha::SetFecha(const int d, const int m, const int a)
{
    dia = d;      // También podría ser Fecha::dia = d;
    mes = m;
    anio = a;
}
```

El uso de este operador no se ciñe exclusivamente a clases. Aunque en este curso será casi su única utilidad.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 25

Introducción a las clases en C++ C++

El operador :: (resolución de ámbito)

Ejemplo de ambigüedad:

```
void Fecha::SetFecha(int dia, int mes, int anio) {
    dia = dia;
    mes = mes;
    anio = anio;
}
```

```
class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(int dia, int mes, int anio);
}; // Se omiten const por falta de espacio
```

Podemos usar el operador de resolución de ámbito:

```
void Fecha::SetFecha(int dia, int mes, int anio)
{
    Fecha::dia = dia;
    Fecha::mes = mes;
    Fecha::anio = anio;
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 26

Introducción a las clases en C++ C++

El puntero this

Siempre que estamos ejecutando un método de una clase, disponemos de un puntero que apunta al objeto que hace la llamada del método: el puntero se llama **this**.

```

void Fecha::SetFecha(const int d, const int m, const int a) {
    this->dia = d;
    this->mes = m;
    this->anio = a;
}

class Fecha {
    ...
};

main() {
    Fecha f;
    ...
    f.SetFecha(6,6,1944);
}

```

Memoria local de SetFecha()

d	[]
m	[]
a	[]
this	[]

Memoria de main()

f	dia	mes	anio
---	-----	-----	------

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 27

Introducción a las clases en C++ C++

La clase Fecha completa

```

class Fecha {
    int dia, mes, anio;
public:
    void SetFecha(const int d, const int m, const int a);
    void GetFecha(int &d, int &m, int &a) const;
};

void Fecha::SetFecha(const int d, const int m, const int a)
{
    dia = d;
    mes = m;
    Fecha::anio = a; // Ejemplo de uso de ::
}

void Fecha::GetFecha(int &d, int &m, int &a) const
{
    d = dia;
    m = mes;
    a = this->anio; // Ejemplo de uso de this
}

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 28

Introducción a las clases en C++ 
La clase Fecha completa

```

void LeeFecha(Fecha &f)
{
    int d, m, a;
    cout << "Dime la fecha: ";
    cin >> d >> m >> a;
    f.SetFecha(d,m,a);
}

int diferencia(const Fecha f1, const Fecha f2)
{
    int f1d, f1m, f1a, f2d, f2m, f2a, d1, d2;
    f1.GetFecha(f1d, f1m, f1a);
    f2.GetFecha(f2d, f2m, f2a);
    d1 = f1d + 30*f1m + 30*12*f1a;
    d2 = f2d + 30*f2m + 30*12*f2a;
    return (d2<d1) ? (-1) : (d2-d1);
}

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  29

Introducción a las clases en C++ 
Métodos inline

Podemos crear métodos **inline** implementándolos, directamente, dentro de la definición de la propia clase.

```

class Fecha {
    int dia, mes, anio;
public:
    inline void SetFecha(const int d, const int m,
                         const int a)
    { dia=d; mes=m; anio=a; }

    void GetFecha(int &d, int &m, int &a) const;
};

```

Opcionalmente, podemos precederlos de la palabra reservada **inline**.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  30

 Metodología de la Programación C++

Tema 5.- Clases y abstracción

Abstracción y encapsulamiento

Introducción a las clases en C++

<i>Atributos y métodos</i>	<i>Partes públicas y privadas</i>
<i>Diferencias con struct</i>	<i>El operador .</i>
<i>Métodos const</i>	<i>Operador :: (resolución de ámbito)</i>
<i>El puntero this</i>	<i>Métodos inline</i>

Funciones y clases amigas

 Universidad de Granada  DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 31

 Funciones y clases amigas C++
¡ Rompiendo el encapsulamiento !

Las funciones y clases amigas (`friend`) pueden acceder a la parte privada de otra clase.

Se suelen usar, puntualmente, por cuestiones justificadas de eficiencia. No es conveniente usarlas indiscriminadamente ya que rompen la abstracción.

```
class A {
    private:
    ...
public:
    ...
    friend class B;
    ...
    friend tipo funcion(parametros);
};
```

B es una clase amiga de A.
 Desde B podemos acceder a la parte privada de A.
`funcion()` es una función amiga de A.
 Desde `funcion()` podemos acceder a la parte privada de A.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 32

Funciones y clases amigas C++ Ejemplo

The diagram illustrates the resolution of friend declarations between two classes, ClaseA and ClaseB, and a free function func().

ClaseA Declaration:

```
class ClaseA {  
private:  
    int x;  
...  
public:  
...  
friend class ClaseB;  
...  
friend void func();  
};
```

ClaseB Declaration:

```
class ClaseB {  
private:  
    ...  
public:  
    ...  
    void unmetodo();  
};  
...  
void ClaseB::unmetodo() {  
    ClaseA v;  
    v.x = 3; // Acceso a v  
    ...  
}
```

func() Declaration:

```
void func() {  
    ClaseA z;  
    z.x = 6; // Acceso a z  
    ...  
}
```

A green arrow points from the friend declaration in ClaseA to the friend declaration in func(). Another green arrow points from the friend declaration in func() to the friend declaration in ClaseB.