

Interfaces

Los Contratos entre Proveedores y Consumidores

Esp. Ing. José María Sola, profesor.

Revisión 4.0.0

2025-06-06

Introducción

En este texto presento los siguientes conceptos y técnicas fundamentales de la programación en general y del Lenguaje C y sus derivados:

- Construcción de abstracciones.
- Dependencia del cliente con respecto a una interfaz, no a una implementación.
- Archivos encabezados como interfaz y guardas de inclusión.
- Proceso de compilación y compilación separada.

Abstracciones e Interfaces

La programación modular es en la construcción de sistemas basadas en **módulos** o **componentes** con una función clara con alta **cohesión** que presentan una interfaz que permite bajar el **acoplamiento** en la comunicación entre módulos.

Un **componente** es una unidad que **provee servicios** a otros componentes, el mecanismo que **implementa** ese servicio es **abstraído** de los componentes mediante una **interfaz pública**.



De esa forma, el componente implementa una abstracción, la cual es provista mediante una interfaz.

La interfaz establece el **contrato** de comunicación, que establece las responsabilidades del **componente proveedor** y del **componente consumidor**.

Al diseñar la interfaz de la abstracción buscamos que nuestros consumidores cumplan el siguiente objetivo:

Depender de la abstracción, no de la implementación.

Para ello la interfaz del componente no debe exponer detalles de implementación, lo cual permite que cambios en el componente no afecten a los consumidores.

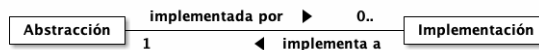
La relación entre el Cliente y la Interfaz puede describirse como que **el cliente importa la interfaz** o también como que **el cliente depende de la interfaz**.

Asimismo, la relación entre la Implementación y la Interfaz puede describirse como que **el proveedor exporta la interfaz** o también como que **el proveedor implementa la interfaz**.

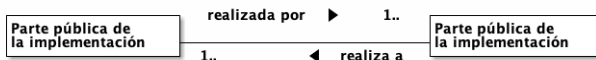


El objetivo final es **construir abstracciones para resolver problemas**. Una abstracción puede implementarse en diferentes lenguajes de programación y de diferentes formas, pero cada implementación siempre tiene:

- una *parte pública o interfaz*, y
- una *parte privada o implementación*.



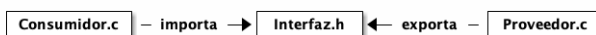
El diseño de la implementación debe permitir cambios en su parte privada, sin requerir cambios en su parte pública.



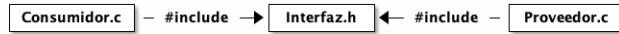
Interfaces en el Lenguaje C y Derivados

En el lenguaje C, y sus derivados, las interfaces se definen en archivos **header** (encabezado), con extensión `.h`, y los consumidores y proveedores en archivos `.c`.

Otras tecnologías aplican los conceptos de forma similar con otros nombres, por ejemplo C# y Java usan `interface` y `class`, y Smalltalk usa `protocol` y clases.



Tanto la relación **importa** como la relación **exporta** en C se realiza con la ayuda de la directiva `#include` del preprocesador.



Si cumplimos la regla que

tanto el consumidor como el proveedor deben incluir `Interfaz.h`.

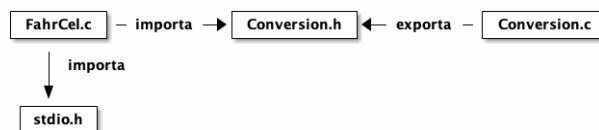
podemos basarnos en el **compilador** para forzar que ambas partes cumplan el contrato. Al ser incluido `Interfaz.h` por ambas partes, el compilador puede detectar los siguientes tipos de errores:

- **Invocación** incorrecta por parte del consumidor.
- **Definición** incorrecta por parte del proveedor.

Como ejemplo, supongamos el caso del programa de conversión de temperaturas de la sección 1.2 y el ejercicio 1-15 de [K&R1988].

Un programa que imprime una tabla de conversión de temperaturas de fahrenheit a celsius depende de un componente que provea el servicio de conversión de forma tal que lo abstraiga de la expresión que implementa la fórmula.

La **abstracción** se logra mediante la función de conversión `double celsius(double fahr)`; la cual se declara en la **interfaz** `conversion.h` y se implementa en el **proveedor** `conversion.c`. El programa que imprime la tabla es `FahrCel.c`, el cual también depende de un mecanismo para enviar datos a la salida estándar, por eso, `FahrCel.c` depende de `conversion.h` y de `stdio.h`.



El contenido de los tres archivos está a continuación:

FahrCel.c.

```
#include <stdio.h>
#include "Conversion.h"

int main(){
    constexpr int LOWER = 0,    // lower limit of table
                UPPER = 300,    // upper limit
                STEP  = 20;     // step size

    for(int fahr = LOWER; fahr <= UPPER; fahr += STEP)
        printf("%3d %6.1f\n", fahr, Celsius(fahr) );
}
```

Conversion.h.

```
#ifndef CONVERSION_H_INCLUDED
#define CONVERSION_H_INCLUDED

double Celsius(double);

#endif
```

Conversion.c.

```
#include "Conversion.h"

double Celsius(double f){
    return (5.0/9.0)*(f-32);
}
```

Y el comando para construir el programa es:

```
cc FahrCel.c Conversion.c -o FahrCel
```

La utilidad `make` [\[SOLA_Make_2022\]](#) permite automatizar la gestión de las dependencias y el proceso de traducción de todo el sistema.

Referencias

[SOLA_Make_2022] José María Sola. *Make: Automatización del Proceso de Traducción* (2022) <https://josemariasola.wordpress.com/ssl/papers#Make>

Changelog

4.0.0+2025-06-06

- Completely removed *Make* section and moved it to paper *Make* for even more cohesion.
- Renamed file and title.

3.0.0+2025-06-05

- Removed *Make* section and moved it to paper *Make* for more cohesion.
- Updated to C23.

2.x.0+2017

- Revamp

1.0.0+2016

- Versión inicial, basada en *Bibliotecas en C*.

