

# Interfaz pública vs Implementación

...

¿Qué son y para qué sirve cada uno?

# Separamos la **interfaz** de la **implementación**

Declaración

**interface**

QUÉ DICE QUE HACE

VS

Definición

**implementación**

CÓMO LO HACE

# Interfaces

- Una interfaz es un conjunto de firmas de métodos.
- Es un contrato entre la persona que va a implementar el TDA (Alan) y la persona que lo va a consumir (Barbara).
- Uno de los objetivos es el de enmascarar implementaciones concretas.

# Interfaces - Sintaxis

```
type nombre interface {  
    metodo1  
    metodo2  
}
```

# Interfaces - Ejemplo

```
type geometry interface {  
    area() float64  
    perim() float64  
}
```

```
type rect struct {  
    ancho, alto float64  
}
```

```
type circulo struct {  
    radio float64  
}
```

```
func (r rect) area() float64 {  
    return r.ancho * r.alto  
}
```

```
func (r rect) perim() float64 {  
    return 2*r.ancho + 2*r.alto  
}
```

```
func (c circle) area() float64 {  
    return math.Pi * c.radio * c.radio  
}
```

```
func (c circle) perim() float64 {  
    return 2 * math.Pi * c.radio  
}
```

# Interfaces - Ejemplo

```
func medidas(g geometry) {  
    fmt.Println(g)  
    fmt.Println(g.area())  
    fmt.Println(g.perim())  
}  
  
func main() {  
    r := rect{ancho: 3, alto: 4}  
    c := circulo{radio: 5}  
  
    medidas(r)  
    medidas(c)  
}
```

# Interfaces

```
g := geometry{radio: 4}
```



Las interfaces NO tienen constructores



# Interfaz

Contiene la **declaración** de las primitivas de nuestro TAD.



- ¿Cuales son los nombres de las primitivas disponibles?
- ¿Qué reciben? ¿Qué devuelven?
- ¿Qué hacen?



# Interfaz

Ejemplo:

*/\* Abre una puerta. Devuelve nada/*

**AbrirPuerta()**

¿Qué hace?

¿Cómo se llama  
la función?

¿Qué recibe?

¿Qué devuelve?

# Interfaz

**Define** para Barbara que primitivas va a poder utilizar

# AbrirPuerta()

¿qué hace por dentro?

¿cómo simula la apertura de una puerta?

¿usa un booleano en True o False para ver si está abierta la puerta?

¿usa un string “abierto” o “cerrado” para guardar el estado de la puerta?

# Implementación

Contiene la **definición** de las funciones de nuestro TAD, es decir, implementa las funciones de la interfaz

```
func (puerta* Puerta) AbrirPuerta {  
    /* código que abre la puerta */  
}
```

# Puerta - lado implementación

```
type Puerta struct {  
    estado int;  
}  
  
func (puerta *Puerta) cerrar () {  
    puerta.estado = 0;  
}
```

```
type Puerta struct {  
    estado bool;  
}  
  
func (puerta *Puerta) cerrar() {  
    puerta.estado = false;  
}
```

# Implementación

¿Qué pasa si tenemos funciones definidas en la implementación pero no en la interfaz?

- Si empiezan con minúscula no se van a exportar
- Si empiezan con mayúscula serían exportadas pero solo se podrían utilizar si se utiliza la implementación **sin la interfaz**

¿Tenemos que incluir la implementación en compilación?

Si, cuando otro archivo quiere usar las funciones de nuestro TAD

# Implementación

- Si la implementación no se exporta no hay forma de utilizarla
- Que vamos a exportar nosotros de nuestra implementación puntual? La **función de creación** que retorna un puntero a la implementación.

# Implementación

- Si la implementación no se exporta no hay forma de utilizarla
- Que vamos a exportar nosotros de nuestra implementación puntual? La **función de creación** que retorna un puntero a la implementación.
- Ej: Exportamos el *CrearCirculo* para que puedan crearse círculos.



# Implementación

```
circulo := CrearCirculo()
```

```
medidas(circulo)
```

```
func medidas(g geometry) {
```

```
    fmt.Println(g)
```

```
}
```

# ¿Es necesario dividir en múltiples archivos mis programas?

NO, si pongo todo en un archivo y anda

Pero... podés leer esto?

[https://github.com/fabiensanglard/vanilla\\_duke3D/blob/master/SRC/ENGINE.C](https://github.com/fabiensanglard/vanilla_duke3D/blob/master/SRC/ENGINE.C)

(8830 líneas de código)

**DON'T BE "THAT GUY"**



**NOBODY  
LIKES "THAT GUY"**

# Generics



Fin

...