The slide features a light gray background with decorative geometric elements in the corners. These include dark gray and light gray shapes with rounded corners, some outlined in black, arranged in a symmetrical pattern. The central text is positioned in the middle of the slide.

## Clase 18

### Delegados y expresiones lambda



# ¿Qué es un DELEGADO?

Un **delegado** es un ***tipo de referencia*** que representa referencias a métodos con una firma particular.

Podemos asociar su instancia con cualquier método que tenga una firma compatible.

A través de la instancia del delegado podemos invocar al método referenciado.

# Delegados vs punteros a función

**No** son lo mismo que un puntero a función:

- Son orientados a objetos.
- Encapsulan tanto la referencia al método como la instancia a la que pertenece.
- Cuentan con seguridad de tipos.

# Usos comunes de los delegados

## **Usar métodos como argumentos**

Los parámetros de entrada de métodos y delegados pueden ser tipos delegados.

*El método Sort de la clase List recibe como argumento un delegado que contiene el criterio de comparación para el ordenamiento.*

## **Almacenar métodos como atributos**

También se pueden declarar atributos y propiedades como tipos delegados, permitiendo que un objeto almacene la referencia a un método de otro objeto para ejecutarlo posteriormente.

## **Devolución de llamada asincrónica (asynchronous callback)**

Se pueden usar para notificar al invocador cuando una tarea prolongada ha terminado.

## **Desacoplar y otorgar flexibilidad**

Son particularmente útiles cuando el código que contiene el método a invocar se desarrolla de manera independiente al código invocador.

# Declaración y uso de un delegado

```
// Declarar un delegado
public delegate void DelegadoNotificacion(string str);

// Método con la misma firma que el delegado.
static void Notificar(string nombre)
{
    Console.WriteLine($"Se recibió una notificación para: {nombre}.");
}
```

# Declaración y uso de un delegado

```
// Instanciar el delegado.  
DelegadoNotificacion delegadoNotificacion = new DelegadoNotificacion(Notificar);
```

```
// Instanciar el delegado  
DelegadoNotificacion delegadoNotificacion = Notificar;
```

```
// Usar el delegado  
delegadoNotificacion("Juan Perez");
```

# Delegados predefinidos

## **Action**

Representa métodos que **no** retornan nada y tienen entre 0 y 16 parámetros de entrada.

## **Func**

Representa métodos que **sí** retornan algo y tienen entre 0 y 16 parámetros de entrada.

## **Predicate**

Representa métodos que retornan **bool** y reciben un parámetro de entrada.

# Delegados multidifusión (multicast)

Una misma instancia de un delegado puede referenciar a varios métodos y llamarlos a todos cuando se accione.

```
public delegate void Delegado(string str);

public class Clase
{
    public void MetodoUno(string str) { }
    public void MetodoDos(string str) { }
    public void MetodoTres(string str) { }
}

class Program
{
    static void Main(string[] args)
    {
        // Instancio el objeto que contiene los métodos que encapsulará el delegado.
        Clase objeto = new Clase();

        // Instancio dos delegados.
        Delegado delegadoUno = objeto.MetodoUno;
        Delegado delegadoDos = objeto.MetodoDos;

        // Ambos tipos de asignación son válidos.
        Delegado delegadoMultidifusion = delegadoUno + delegadoDos;
        delegadoMultidifusion += objeto.MetodoTres;
    }
}
```





# ¿Qué es una EXPRESIÓN LAMBDA?

Las **expresiones lambda** son funciones con una sintaxis diferente que permite utilizarlas en una expresión en lugar de los típicos métodos que son miembros de una clase.

Permiten crear funciones anónimas. Una **función anónima** es una función que no tiene nombre ni declaración formal.

# Sintaxis de una expresión lambda

El **operador lambda** (**=>**) se utiliza para separar la lista de parámetros del cuerpo del método anónimo.

Los parámetros de entrada van a la izquierda del operador lambda y la sentencia o bloque de instrucciones al otro lado.

*Lambda operator*

*Parameter*    *s => s.Age > 12 && s.Age < 20;*    *Body Expression*

# Sintaxis de una expresión lambda

## **Expresión lambda**

(parámetros de entrada)  $\Rightarrow$  expresión

```
int CalcularPotenciaAlCuadrado (int numero)
{
    return numero * numero;
}
```

```
n  $\Rightarrow$  n * n
```

# Sintaxis de una expresión lambda

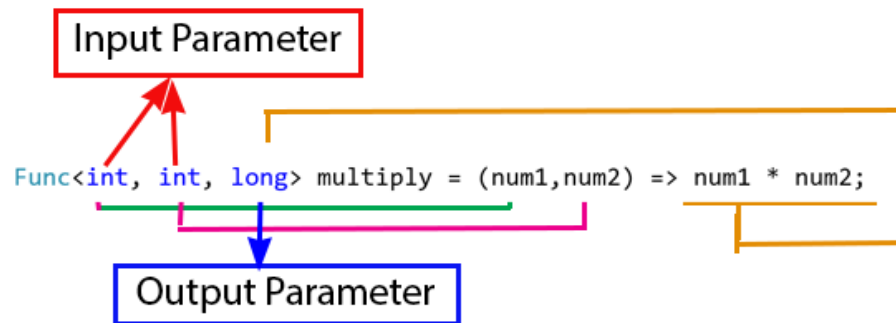
## **Instrucción lambda**

(parámetros de entrada) => { bloque de sentencias }

```
void Saludar (string nombre, string apellido)
{
    string saludo = $"¡Bienvenido {nombre} {apellido}!";
    Console.WriteLine(saludo);
}
```

```
(n, a) => {
    string saludo = $"¡Bienvenido {nombre} {apellido}!";
    Console.WriteLine(saludo);
}
```

# Delegados y expresiones lambda



```
Action<string> saludar = nombre =>
{
    string saludo = $"¡Hola {nombre}!";
    Console.WriteLine(saludo);
};

saludar("Mundo");
```

```
Func<int, int> elevarAlCuadrado = x => x * x;

Console.WriteLine(elevarAlCuadrado(5));
```