TIPOS

Los objetos colaboran enviándose mensajes para consultar o cambiar el estado y activar comportamientos. El objeto emisor debe solicitar solo lo que el receptor está diseñado para manejar, mientras que el receptor debe cumplir con lo que está especificado en el contrato.

Este contrato define los derechos y obligaciones de cada objeto, estableciendo qué puede solicitar el emisor y qué debe hacer el receptor. Los tipos definen estos contratos a través de operaciones, cada una con una firma que incluye nombre, argumentos y resultado.

TIPO:

- Un **tipo** es un conjunto de **operaciones** que determina los mensajes que pueden ser enviados a los objetos de ese tipo.

FIRMA:

- La **firma** está compuesta por el **nombre**, los **parámetros** y **tipo de los parámetros**, y el **tipo del resultado** de la operación.

No hay firmas duplicadas.

El uso de tipos permite a los objetos conocer qué operaciones pueden solicitar y cómo hacerlo. Una clase que implementa un tipo tiene métodos que coinciden exactamente con las operaciones del tipo, y se dice que la clase implementa ese tipo.

CLASES IMPLEMENTAN TIPOS:

- La **clase** de un objeto **implementa** un **tipo** si tiene un método para cada una de las operaciones incluidas en el tipo.
- La firma de cada método coincide con la de la operación correspondiente.

MENSAJES Y MÉTODOS:

- El método que se ejecuta como consecuencia de la recepción de un mensaje tiene el mismo nombre que el selector del mensaje, los mismos parámetros que el mensaje, y los parámetros son del mismo tipo que los del mensaje.

Cuando un objeto envía un mensaje a otro, el mensaje dice quién es el objeto **receptor**, cuál es el nombre de la operación,, y cuáles son los valores de los argumentos, si los hubiere.

EMISORES Y TIPOS:

- El emisor de un mensaje está obligado a usar las operaciones incluidas en el tipo del receptor y tiene el derecho de usar cualquier de las operaciones incluidas en el tipo.

RECEPTORES Y TIPOS:

- El receptor de un mensaje está obligado a que su clase tenga un método para cada operación incluida en su tipo y tiene el derecho de que su clase

tenga métodos sólo para esas operaciones y ninguna más.

> Un objeto puede colaborar con muchos otros, requiriendo contratos diferentes para cada colaboración, lo que significa que puede tener varios tipos. Además, diferentes objetos, incluso de distintas clases, pueden compartir el mismo tipo si prestan los mismos servicios de manera idéntica.

UN OBJETO, MAS DE UN TIPO:

- Un objeto puede tener más de un tipo si la clase de ese objeto implementa más de un tipo.

UN TIPO, MAS DE UN OBJETO:

- Objetos de clases diferentes pueden tener el mismo tipo si las clases de esos objetos implementan el mismo tipo.

DECLARACION DE TIPOS:

- Los tipos pueden ser declarados explícitamente usando **interfaces** e implícitamente usando **clases**.

ABSTRACCION:

- Una **abstracción** expresa las características **esenciales** de un objeto, que lo distinguen de todos los demás tipos de objetos, y que provee límites conceptuales claramente definidos, relativos a la perspectiva del usuario.

El objeto receptor de un mensaje no necesita saber quién envía el mensaje, solo debe cumplir con el tipo especificado. Esto simplifica la colaboración, ya que el emisor solo necesita conocer el tipo del receptor y no detalles adicionales. Un objeto puede aceptar mensajes si tiene las operaciones del tipo requerido, incluso si tiene más. Un tipo es subtipo de otro si incluye todas sus operaciones, y es supertipo si sus operaciones están incluidas en el otro.

SUBTIPO:

- Un tipo A es **subtipo** de otro B si el conjunto de operaciones del tipo A **contiene** el conjunto de las del tipo B.
- El **subtipo** A suele ser **menos abstracto** o **más concreto** que el tipo B.

SUPERTIPO:

- Un tipo A es **supertipo** de otro B si el conjunto de operaciones del tipo A **está contenido** en el conjunto de las del tipo B.

> Lo anterior se resumen en el **principio de sustitución**: en cualquier contexto en el que sea válido usar un objeto con un cierto tipo, se puede sustituir ese objeto por otro con un subtipo de ese tipo, porque puede recibir y procesar exactamente los mismos mensajes.

PRINCIPIO DE SUSTITUCION:

- En cualquier lugar de un programa donde se espera un objeto de un tipo puede aparecer un objeto de un subtipo y el comportamiento del programa no debería cambiar.

POLIMORFISMO:

- El **polimorfismo** permite que una misma definición pueda ser usada con diferentes tipos.

OPERACION POLIMÓRFICA:

- Una operación es polimórfica si puede ser usada con diferentes tipos, incluyendo sus subtipos, permitiendo que objetos de distintos tipos respondan al mismo mensaje de manera intercambiable. También es polimórfica si los argumentos y el resultado pueden variar con diferentes tipos, lo que se conoce como sobrecarga.

SOBRECARGA:

- Una operación está sobrecargado cuando hay múltiples versiones con el mismo nombre, pero con parámetros diferentes, tipos de parámetros distintos, o resultados distintos.

> Las clases, especifican cómo se implementan los tipos, incluyendo métodos y atributos.
>
> En C#, los tipos se definen mediante interfaces, y las clases implementan estos tipos al proporcionar el código necesario para manejar los mensajes especificados.

NOTA:

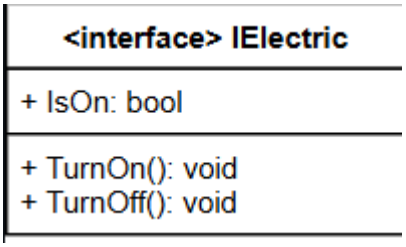
> Cuando una clase implementa un tipo declarado mediante la palabra clave **interface**, tiene en realidad dos tipos: el tipo explícito de la declaración y el tipo implícito compuesto por todos los métodos y atributos de esa clase.

Definiremos los tipos **IElectric** para las cosas que puedo prender y apagar y **ILikeable** para las que me pueden gustar o no, explícitamente como interfases; también definiremos una clase **Car** que implementará los tipos **IElectric** y **ILikable** y por lo tanto tendrá los tipos **Car**, **IElectric** y **ILikeable** y una clase Actor que implementará el

tipo **ILikeable** y por lo tanto tendrá los tipos **Actor** y **ILikeable**.

```
using System;

public interface IElectric
{
    Boolean IsOn { get; }
    void TurnOn();
    void TurnOff();
}
```

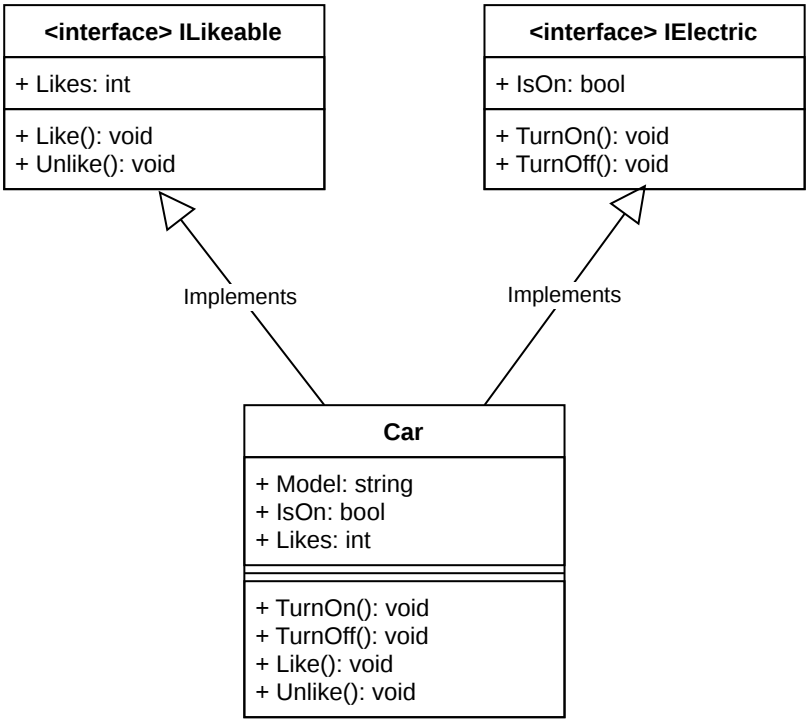
- En UML representamos las interfaces de forma similar a las clases, pero agregando `<interface>` antes del nombre de la interfaz.




```
using System;

public class Car : IElectric, ILikeable
{
    private String model;
    private Boolean isOn;
    private Int32 likes;

    public Car(String model)
    {
        this.model = model;
        this.isOn = false;
        this.likes = 0;
```

```
    }

    public String Model
    {
        get
        {
            return this.model;
        }
    }

    public Boolean IsOn
    {
        get
        {
            return this.isOn;
        }
    }

    public Int32 Likes
    {
        get
        {
            return this.likes;
        }
    }

    public void TurnOn()
    {
        this.isOn = true;
    }

    public void TurnOff()
    {
        this.isOn = false;
    }

    public void Like()
    {
```

```
        this.likes = this.likes + 1;
    }

    public void Unlike()
    {
        if (this.likes > 0)
        {
            this.likes = this.likes - 1;
        }
    }
}
```



PRINCIPIO DE SUSTITUCION EN LA REDACCION DE LISKOV:

- Si para cada objeto O de tipo S existe un objeto O' de tipo T tal que para todos los programas P definidos en términos de T, el comportamiento de P permanece sin cambios cuando O es substituido por O', entonces S es un subtipo de T.

📄 TIPOS GENÉRICOS:

📌 TIPOS GENÉRICOS (C# GUIDE)