RESUMEN BÁSICO: RELACIONES ENTRE OBJETOS

(COMPOSICIÓN Y AGREGACIÓN)

1. Introducción

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí.

Existen diferentes tipos de relaciones, entre las cuales nos vamos a centrar en este documento en el conocimiento de la **composición** y la **agregación**, que son dos tipos de relaciones de asociación que se pueden establecer entre objetos. El conocimiento de este tipo de relaciones será la base para afrontar la siguiente actividad de ejercicios prácticos sobre POO (AP10)

En resúmenes posteriores trataremos otro tipo de relación, como es la **herencia**, pilar fundamental en el paradigma de la POO.

Muchos programadores no tienen clara la diferencia entre composición y agregación, e incluso confunden sus términos. También hay quien confunde agregación con asociación. Intentemos aclarar de una forma breve y rápida cada término y cómo podemos implementar de una forma básica dichas relaciones utilizando C#.

De forma general, y a modo resumen:

- La **Herencia** nos dice o establece una relación "**es un**".
- La **Composición** nos dice o establece una relación "**es parte de**".
- La **Agregación** nos dice o establece una relación "**tiene un**".

Tanto la **Agregación** como la **Composición**, son dos tipos especiales de la **Asociación**.

2. Composición

La **Composición**, es una relación más fuerte que la Agregación.

La Composición se suele representar en UML con un rombo de color negro en un extremo de las clases.

Aquí, estaríamos diciendo que un objeto de la clase Hotel está compuesto por uno o varios objetos de tipo Habitación.

Por otro lado, también estamos diciendo que la clase Habitación es parte de la clase Hotel.

En esta relación, no tiene sentido que un objeto Habitación viva de forma independiente sin formar parte de un objeto Hotel.

También diremos que al eliminar un objeto de la clase Hotel eliminaremos, por lo tanto, los objetos de la clase Habitación que forman parte de él, ya que la relación entre ambas clases es estrecha.

A nivel de código, **la composición implica que un objeto está instanciando a otros objetos dentro de su propio código**. Como regla general, si dentro del código de un objeto creas una instancia de otro (usando new), entonces el primer objeto está compuesto por el segundo.

Un ejemplo en código podría ser el siguiente:

Clase Habitación

```
class Habitacion
    {
        private int numero;  // Número de la habitación.
        private bool reservada = false;  // No reservada inicialmente

        // Propiedades para consultar número de habitación y estado de reserva
        public int Numero { get { return numero; } }
        public bool Reservada { get { return reservada; } }

        // Constructor que asigna el número a la habitación
        public Habitacion(int num)
        {
            numero = num;
        }

        // método para reservar la habitación
        public void Reservar()
        {
            reservada = true;
        }

    }
```

Clase Hotel

```
class Hotel
    {
        private string nombreHotel;     // nombre del hotel
        private int estrellas;          // estrella del hotel
        //Lista de habitaciones
        private List<Habitacion> rooms = new List<Habitacion>();

        // Propiedad para consultar la lista de habitaciones
        public List<Habitacion> Rooms { get { return rooms; } }

        // Constructor del Hotel con nombre, estrellas y número de habitaciones.
        public Hotel(string nom, int est, int numHab)
        {
            // El hotel se crea con numHab habitaciones.
            for (int i = 1; i <= numHab; i++)
            {
                rooms.Add(new Habitacion(i));
            }
            // Nombre y estrellas
            nombreHotel = nom;
            estrellas = est;
        }
```

```
        // Método para acceder al nombre del hotel.
        // Podría hacerse con una Propiedad.
        public string getNombreHotel ()
        {
            return nombreHotel;
        }

    }
```

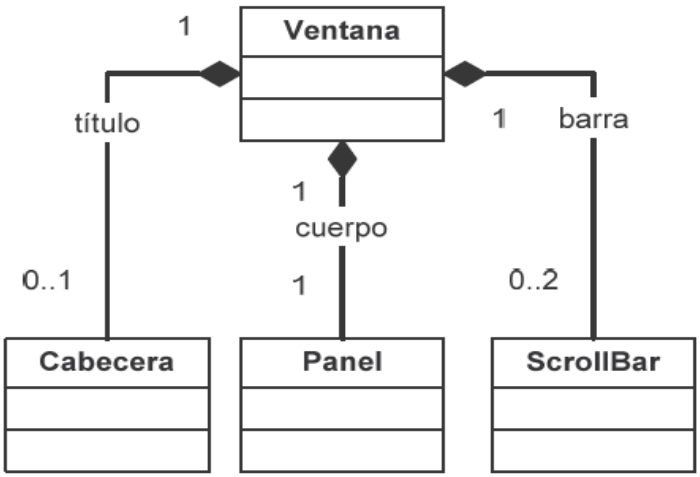
Un ejemplo básico de utilización de un objeto de la clase Hotel desde la clase Program y de gestión de sus habitaciones sería el siguiente:

```
class Program
    {
        static void Main(string[] args)
        {
            // Creamos el hotel Florida Palace de 5 estrellas y 3 habitaciones
            Hotel florida = new Hotel("Florida Palace", 5, 3);

            // Reservamos la primera habitación del hotel
            florida.Rooms[0].Reservar();

            // Vemos el estado de reservas/ocupación de las habitaciones.
            Console.WriteLine("Hotel {0}", florida.getNombreHotel());
            foreach (Habitacion hab in florida.Rooms)
            {
              Console.WriteLine("Habitación: {0} --> Reservada: {1}", hab.Numero,
              hab.Reservada);
            }
            Console.ReadKey();
        }
    }
```

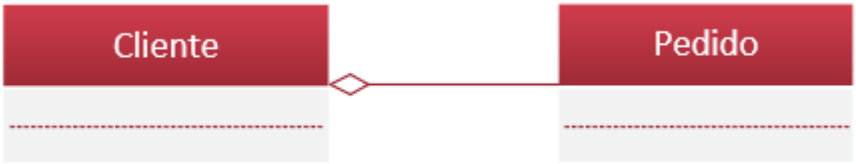
Otros ejemplos de relaciones de Composición podrían ser los siguientes:



Un objeto de la clase Ventana contiene y se compone de tres objetos (cabecera, panel y scrollBar). Puede ocurrir que la ventana no tenga cabecera ni barras de desplazamiento –ScrollBar-, pero sí deberá tener un panel, y el máximo de barras de desplazamiento podrán ser dos (vertical y horizontal). Al ser una relación de Composición, lo que sí debe quedar claro es que en caso de que el objeto de la Clase Ventana se destruyese, también se destruirían sus objetos contenidos de tipo Cabecera, Panel y ScrollBar.

3. Agregación

La Agregación se suele representar en UML con un rombo de color transparente en un extremo de las clases.



La Agregación tiene una relación más débil que la Composición.

Tanto es así, que cuando eliminamos la clase Cliente, no tenemos por qué eliminar la clase Pedido, de manera tal que la clase Pedido podría continuar existiendo.

Aquí, un Cliente agrupa 0, 1 o muchos Pedidos.

La Agregación implica que un objeto A tiene como atributos a otros objetos B, C, etc.., que fueron pasados como parámetros y que no fueron instanciados dentro del objeto A. En todo caso se mantiene una referencia a los objetos B, C, etc… pero en el momento de ser destruido el objeto A, los objetos B y C sobreviven ya que lo único que se pierde es la referencia a los mismos. El objeto A solamente guarda las referencias a los objetos B y C después de haberlas recibido como parámetros.

Llevando el ejemplo de relación anterior (Cliente – Pedido) a código en C#, un ejemplo básico podría ser el siguiente:

Clase Pedido

```
class Pedido
    {
        private int numPedido;              // Número de Pedido
        private string nombreCliente;       // Nombre del Cliente
        private string nombreProducto;      // Nombre del producto pedido
        private double cantidad;            // Unidades pedidas del producto

        // Constructor de pedido con número, nombre producto y unidades producto.
        public Pedido (int num, string nomPro, int cant)
        {
            numPedido = num;
            nombreProducto = nomPro;
            cantidad = cant;
            nombreCliente = "";  // todavía no está asignado a un cliente
        }

        // Método de acceso para la lectura del número de pedido
        public int getNumPedido ()
        {
            return numPedido;
        }

        // Método de acceso para asignar el nombre del cliente que hace el pedido
        public void setNombreCliente(string nom)
        {
            nombreCliente = nom;
        }

        // Método para consultar el nombre del cliente al que pertenece el pedido
```

```
        public string getNombreCliente()
        {
            return nombreCliente;
        }

        // Método para visualizar los datos del pedido.
        public void VerPedido ()
        {
              Console.WriteLine("\nNumero Pedido: {0} // Cliente: {1}", numPedido,
              nombreCliente);
              Console.WriteLine("Producto: {0} // Cantidad: {1}\n", nombreProducto,
              cantidad);
        }
    }
```

Clase Cliente

```
class Cliente
    {
        private string nombre;          // Nombre del cliente
        private string apellidos;       // Apellidos del cliente
        private string nif;             // nif del cliente
        private ArrayList encargo;      // Lista de pedidos asociados al cliente

        // Propiedades para consultar el nombre, apellidos y nif del cliente
        public string Nombre { get { return nombre; } }
        public string Apellidos { get { return apellidos; } }
        public string NIF { get { return nif; } }

        // Constructor de un cliente con nombre, apellido y nif.
        public Cliente(string nom, string ape, string n)
        {
            nombre = nom;
            apellidos = ape;
            nif = n;
            encargo = new ArrayList();  // Instancio la lista de pedidos.
                                        // Guardará referencias a objetos Pedido.
        }

        // Método para realizar/asignar un Pedido al cliente.
        public int RealizaPedido (Pedido orden)
        {
            // el pedido se asigna al cliente (suministrándole su nombre)
            orden.setNombreCliente(nombre);
            // Se incluye en la lista de pedidos del cliente
            encargo.Add(orden);
            // devuelve el número del pedido asignado.
            return orden.getNumPedido();
        }

        // Método para visualizar los pedidos de un cliente.
         public void VerPedidos()
        {
            foreach (Pedido ped in encargo)
            {
                ped.VerPedido();
            }
        }
    }
```

Un ejemplo básico de utilización de objetos Cliente y Pedido desde la clase Program sería el siguiente:

```
class Program
    {
        static void Main(string[] args)
        {
            // Creamos un cliente
            Cliente cli1= new Cliente("Juan", "Marques", "75889654F");

            // Creamos un pedido con número de identificación 10 de dos mesas
            Pedido ped1 = new Pedido(10, "mesa", 2);
            // Creamos un pedido con número de identificación 11 de 4 sillas
            Pedido ped2 = new Pedido(11, "silla", 4);
            // Creamos un pedido con número de identificación 12 de 6 puertas
            Pedido ped3 = new Pedido(13, "puertas", 6);

            cli1.RealizaPedido(ped1); // Se le asigna el pedido ped1 al cliente
            cli1.RealizaPedido(ped3); // Se le asigna el pedido ped3 al cliente

            // Visualizamos los pedidos del cliente
            cli1.VerPedidos();   // Visualizará el ped1 y el ped3.
            Console.ReadKey();
        }
    }
```

En el anterior ejemplo, se crean tres pedidos (ped1, ped2 y ped3) y un cliente (cli1). Al cliente (cli1) se le asignan los pedidos (ped1 y ped2) mediante su método RealizaPedido y posteriormente, desde el cliente se visualizan todos sus pedidos. Los pedidos son objetos no contenidos en el cliente, por lo que si se destruye el objeto cliente dichos pedidos seguirán existiendo.

4. **Anexo** (Ejemplo final y videos de apoyo)

Como resumen, en el siguiente ejemplo:

```
namespace Resumen
{
    class Movil
    {
        Mensaje msg = new Mensaje();
        Software soft = new Software();
        Camara cam = new Camara();
        Bateria bat = null;
        TarjetaSim sim = null;
        TarjetaMemoria memo = null;

        public Movil (Bateria b, TarjetaSim s, TarjetaMemoria m)
        {
            this.bat = b;
            this.sim = s;
            this.memo = m;
        }
    }

    class Mensaje { }
    class Software { }
    class Camara { }
    class Bateria { }
    class TarjetaSim { }
    class TarjetaMemoria { }
}
```

Si se destruye un objeto de la clase Móvil, los mensajes, el software y la cámara también se destruyen (el móvil se compone de ellos), pero la batería, las tarjetas SIM y de memoria perduran (no se destruyen), ya que están agregadas.

Videotutoriales de apoyo:

Programando la Agregación:

¿Cómo crear e implementar Agregación con C#?

Programando la Composición:

¿Cómo crear e implementar Composición con C#?

¡¡ Es necesario visualizarlos!!