Relaciones entre clases

Asociación

Relaciones entre clases

Como hemos visto, los objetos no existen de forma aislada en un sistema sino que deben relacionarse entre sí para ejecutar determinadas tareas.

Es en el diagrama de clases donde se define como serán esas relaciones.

Hay varios tipos de relación entre clases, y definir cual es la que debo utilizar depende de contexto del problema: Asociación, Dependencia, Agregación, Composición y Herencia. Nos concentraremos ahora en la **Relación de asociación.**

Relación de asociación

- Es la forma más simple de relación entre dos clases; se indica cuando un objeto "conoce" a otro objeto y "usa" servicios u operaciones brindados por el otro.
- Ejemplo una persona vive en una casa.

Relación entre clases

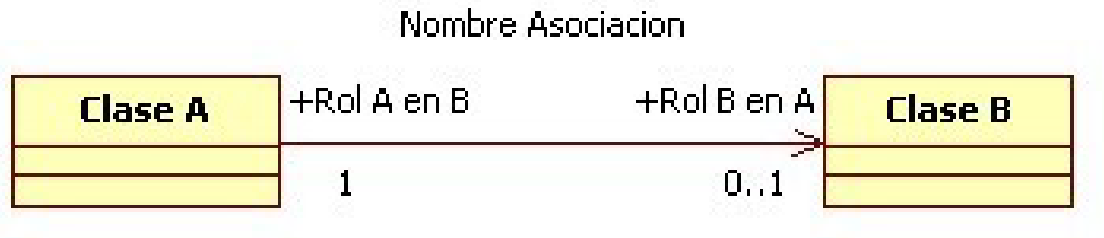
Las asociaciones pueden tener :

Roles

Nombre

Multiplicidad o Cardinalidad

Navegabilidad

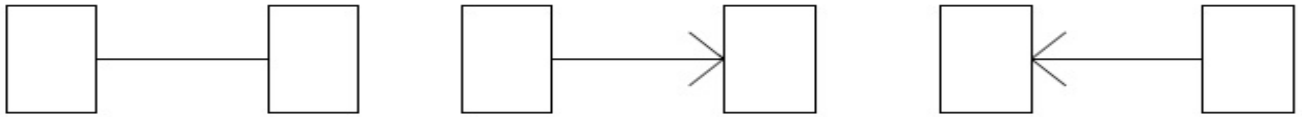


Navegabilidad

Establece quien debe conocer a quien en la relación.

Dos tipos: unidireccional y bidireccional.

La ausencia de símbolo en ambos extremos indica Bidireccionalidad.

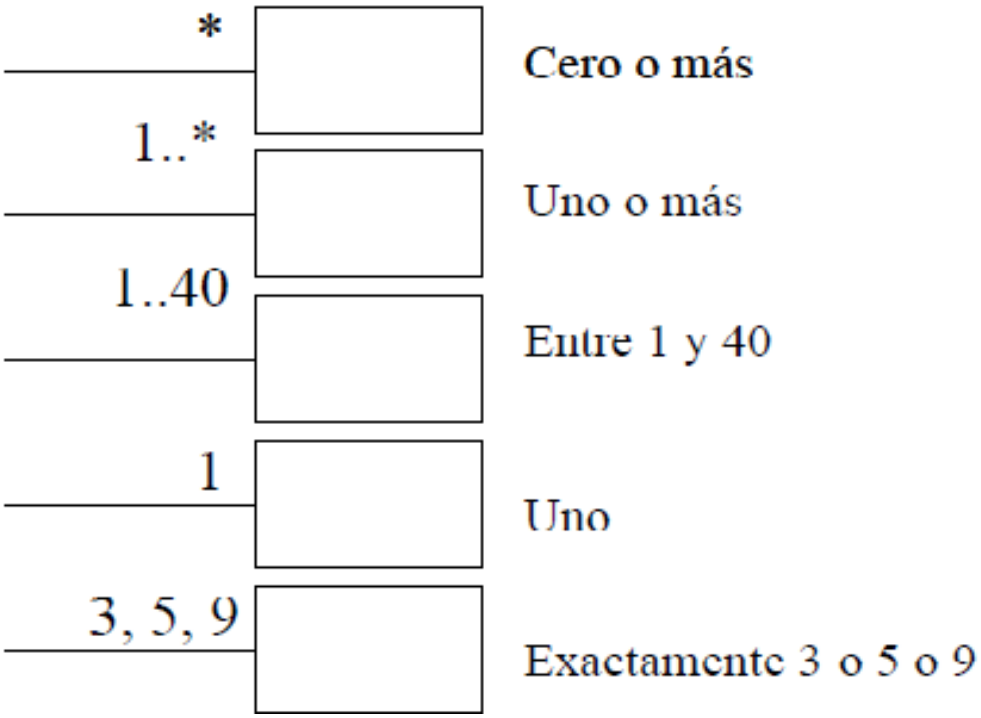


Bidireccional: La persona conoce a su mascota
y la mascota conoce a su dueño

Unidireccional: La persona conoce a su mascota
y la mascota NO conoce a su dueño

Multiplicidad

Indica cuantos objetos de una clase se conectan con cuántos de otra clase.



Implementación de la Asociación

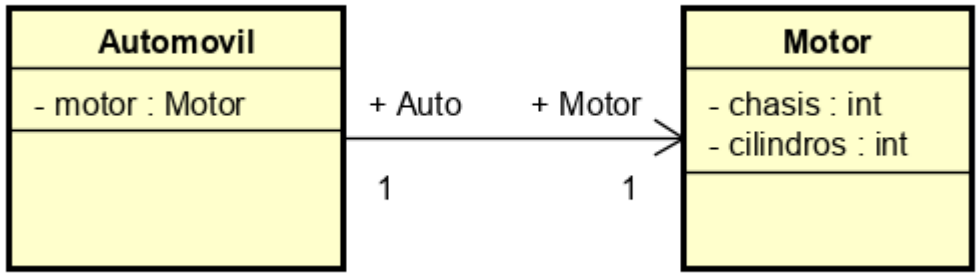
- En todos los casos las asociaciones se implementan mediante referencias (punteros) a objetos.
- Si la relación es de 1 a N o N a N se implementan mediante colecciones de referencias (listas).

Implementación - Asociación 1 a 1 unidireccional

El automóvil conoce al motor, el motor no conoce al automóvil.

Para ello se usa la navegación desde el Automóvil y el Motor.

.



Implementación - Asociación 1 a 1 unidireccional

Se implementa: En la clase Automóvil hay una referencia a una instancia de un Motor, y en la clase Motor **NO** hay una referencia a una instancia de Automóvil. El motor **NO** conoce a su automóvil.

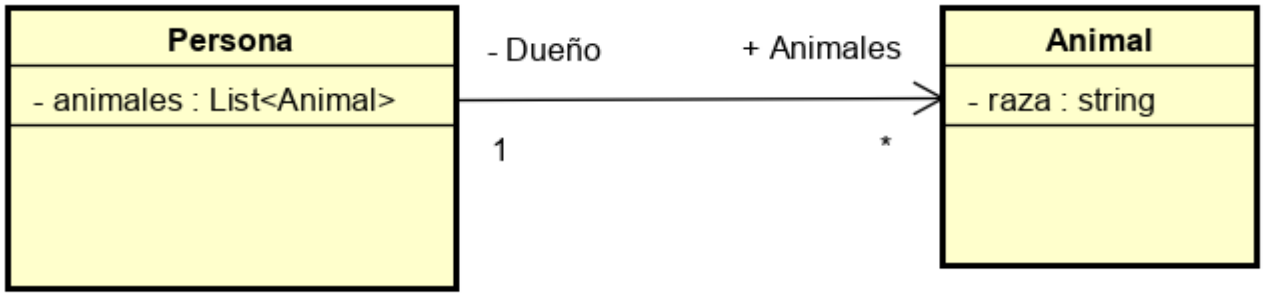
```
public class Automovil
{
    private Motor _motor;

}
```

Implementación - Asociación 1 a muchos unidireccional

Una persona conoce a sus mascotas, y cada animal posee un único dueño.

Para ello se usa la navegación desde el Persona hacia Animal.



Implementación - Asociación 1 a muchos unidireccional

Se implementa: Cada Persona una colección de sus mascotas; en animal no hay referencia a su dueño.

```
public class Persona
{
    private List<Animal> _animales = new List<Animal>();
    //otra opcion es crear la instancia en el constructor de la clase

    0 referencias
    public List<Animal> Animales { get; }
    //el metodo set no se implementa ya que se debe realizar un
    //metodo personalizado para agregar a la lista
    //de esta forma nos aseguramos que se cumplen las reglas de negocio
}
```

Tener en cuenta que si la multiplicidad es 1 a muchos, al instanciar un objeto persona, necesariamente debe recibir en el constructor a su primera mascota, en caso de no ser necesario, la multiplicidad debe ser cero a muchos o simplemente *

Implementación - Asociación 1 a muchos unidirecional

Para agregar un nuevo animal: Se realiza un método en la clase Persona que es la que conoce la lista de animales, el método controla que cumpla con la regla del negocio (la raza no puede estar vacía) y la agrega a la lista.

```
// en clase Animal
public void Validar()
{
    if (raza.Length == 0)
        throw new Exception();
}
```

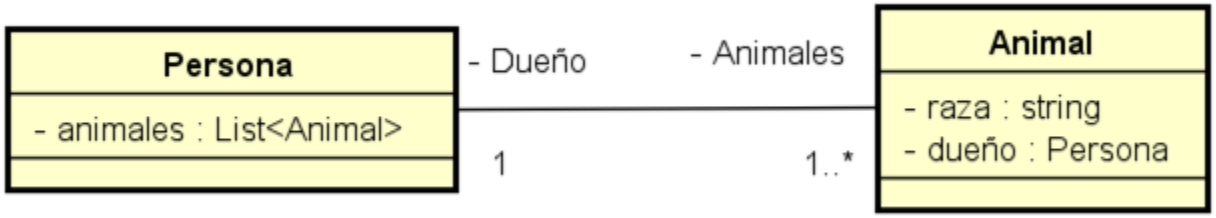
```
// en clase Persona
// Es importante hacer los controles necesarios
// según las reglas de negocio solicitadas.
public void AgregarAnimal(Animal unAnimal)
{
    try
    {
        unAnimal.Validar();
        animales.Add(unAnimal);
    }
    catch (Exception e)
    {
        throw e;
    }
}
```

Implementación - Asociación 1 a muchos bidireccional

Una persona conoce a sus mascotas, y cada animal conoce a su dueño, y ambos se “conocen”

Para ello se usa la navegación bidireccional

Las relaciones bidireccionales deben evitarse si no son estrictamente necesarias, aunque dependiendo del contexto del problema puede haber situaciones en donde sea necesario utilizarlas.



Implementación - Asociación 1 a muchos bidireccional

```
public class Persona
{
    private List<Animal> _animales = new List<Animal>();
    0 referencias
    public Persona(Animal unaMascota)
    {
        try
        {
            AgregarAnimal(unaMascota);
            unaMascota.Duenio = this;
        }catch(Exception e)
        {
            throw e;
        }
    }
    1 referencia
    public void AgregarAnimal(Animal unaMascota)...
}
```

```
public class Animal
{
    private string _raza;

    private Persona _duenio;

    0 referencias
    public string Raza
    {
        get { return _raza; }
        set { _raza = value; }
    }
    1 referencia
    public Persona Duenio
    {
        get { return _duenio; }
        set { _duenio = value; }
    }
    0 referencias
    public void Validar()...

}
```

Relaciones entre clases

Relación de Dependencia

- Es una relación semántica entre dos elementos en la cual un cambio a un elemento (elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente).
- Un cambio en la clase independiente puede afectar a la clase dependiente.
- Se representa con una línea discontinua.
- Un ejemplo es el formulario de personas, depende de la clase persona