UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

UML

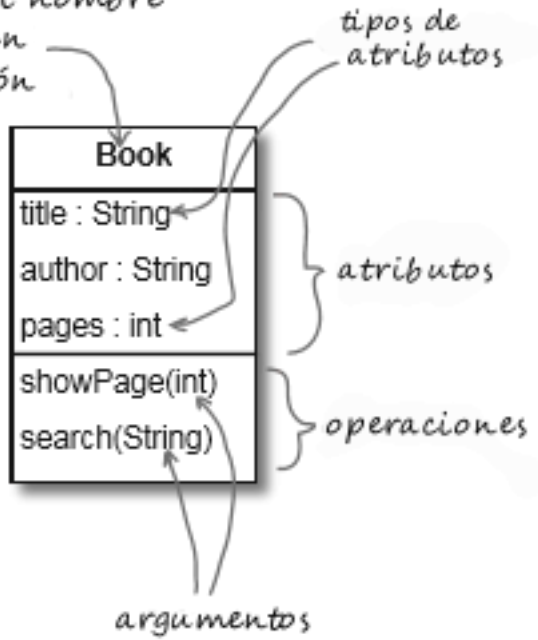
UML es el acrónimo de Unified Modeling Language, y es la notación mas usada para crear diagramas que describen sistemas orientados a objetos. Nuestra primera mirada a UML, será a la notación utilizada para representar clases (probablemente el aspecto mas usado de UML) como así también a la notación utilizada para representar objetos individuales.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Clases y Objetos

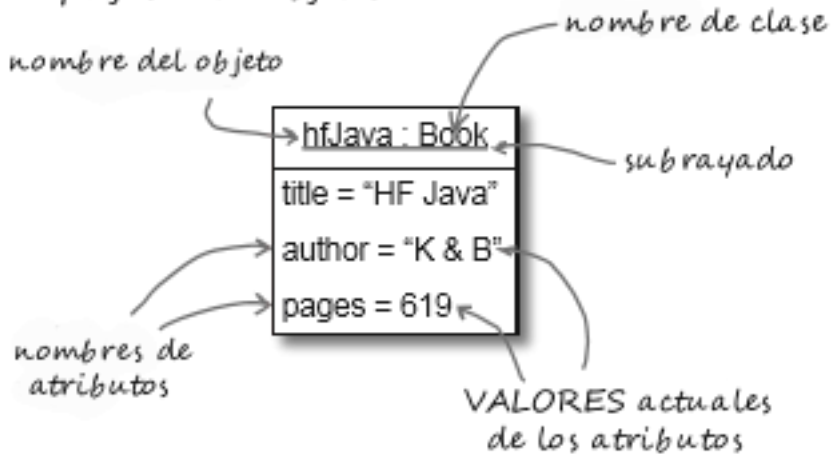
Clases

Una clase es dibujada como un rectangulo con tres secciones. El nombre de la clase va en la primer sección



Objetos

Es menos común dibujar diagrams de objetos que diagramas de clases. Pero pueden ser muy utiles para describir ejemplos específicos o relaciones complejas entre objetos.

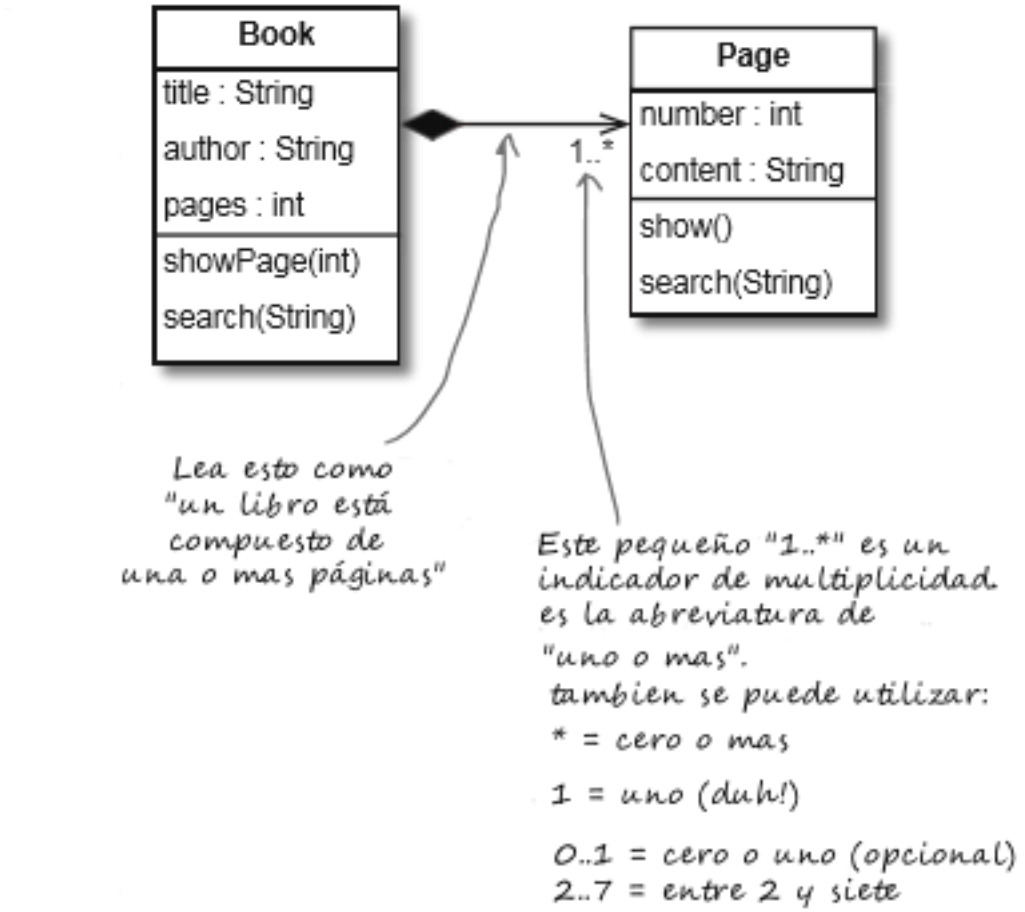


Note que no nos preocupamos por listar las operaciones cuando dibujamos una instancia de un objeto. Todos los objetos de un tipo dado (clase) tienen el mismo conjunto de operaciones, por lo que seria redundante listarlas aquí.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Asociaciones

Las asociaciones son dibujadas en UML conectando dos clases con una linea. Dependiendo del tipo de relacion, puede haber un diamante en uno de los finales de la linea. En este ejemplo, el diamante es lleno, lo que significa que estamos mostrando un tipo especial de asociación llamado "composición"



UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Asociaciones

Composición vs. Asociación

El pequeño diamante indica una relación de composición. La forma mas fácil de pensar acerca de la relación de composición es como una relación “es-parte-de”: Una pagina es parte de un libro. Si usted observa el ejemplo Libro/Página (el cual usa composición), el punto clave aquí es si el libro desaparece, todas las páginas tambien desaparecen! El libro esta compuesto de páginas.

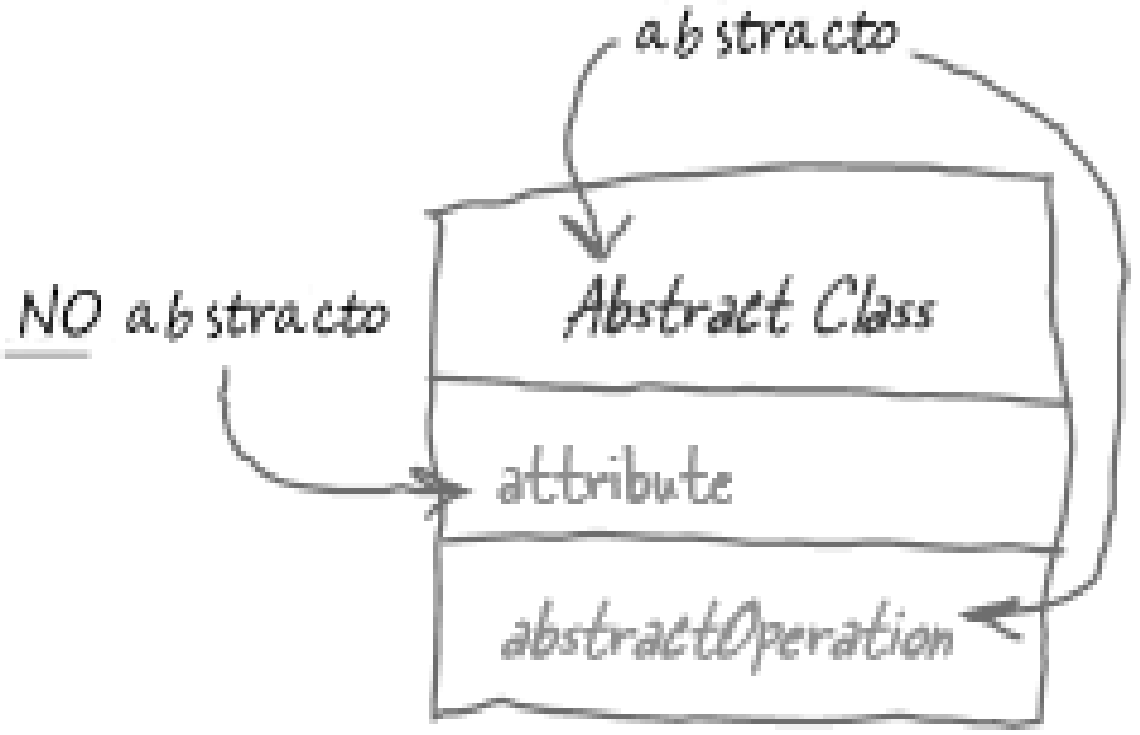
Navegación

Otro punto clave para entender la notación de las relaciones en UML es la navegación Si una linea entre dos clases tiene una flecha en un final de esta(no un triangulo), esto nos está diciendo que la relación puede solo ser transversa en una dirección En el ejemplo, estamos diciendo que un Libro conoce sus páginas, pero una página no sabe en que libro está.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Cosas abstractas

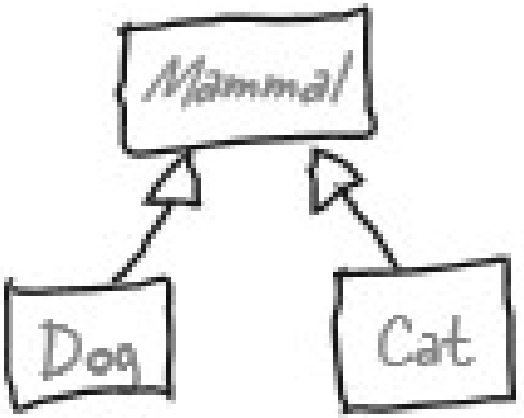
Todo lo que sea abstracto en un diagrama UML es escrito utilizando *cursiva*.



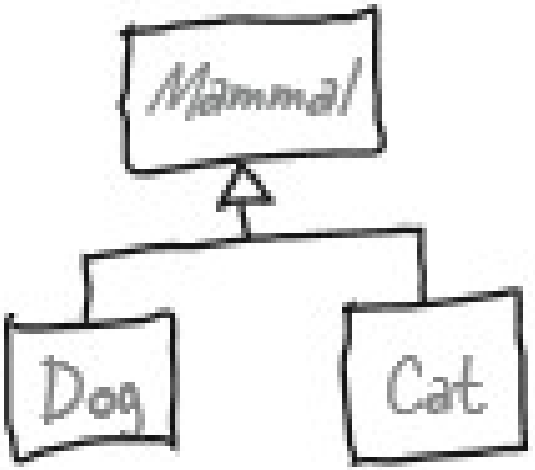
UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Generalización

se muestra en UML utilizando un triangulo abierto que apunta desde la clase mas especifica a la clase mas general



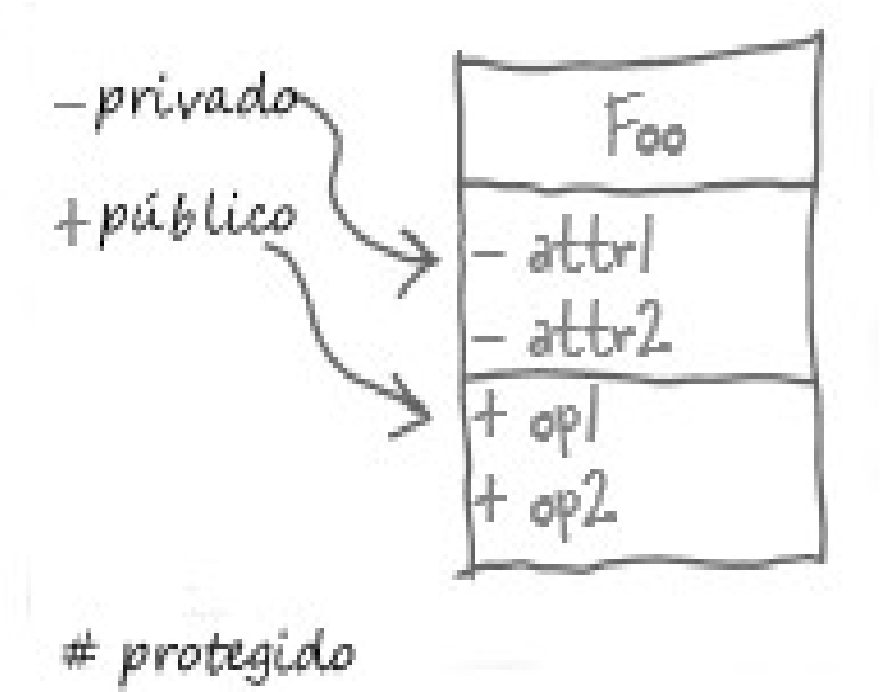
Si Ud. prefiere, tambien es válido en UML usar un solo triangulo con múltiples lineas conectadas a él, de esta forma:



UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Indicador de visibilidad

Ud. puede indicar cuando un atributo u operación es público, privado o protegido precediendolo con un indicador de visibilidad (o modificador de acceso). Los indicadores son:



UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Operaciones

Se muestran en la sección de abajo de la caja de la clase. Las operaciones se especifican de la siguiente forma:

```
Mod_acceso nombre(nombre_param:tipo_param, …) : tipo_retorno
```

Si tienes una operación pública que suma dos números, se vería así:

```
+ add(a:int, b:int) : int
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Roles

Los roles nos dicen que trabajo cumple cada clase en una asociación
Ademas de las multiplicidades, ud. puede agregar “roles” que sirven para nombrar las asociaciones en casos donde puede no ser claro que rol juega una clase en una asociación particular. Agregando roles a cada asociación puede ser tedioso. Ud. debe agregarlos solo si ayudan a clarificar el diagrama. Los nombres de Rol son escritos en el misma área general de las multiplicidades. Asegúrese de poner el nombre del rol junto a la clase que representa. Por ejemplo:

Roles

Aqui, tenemos 2 asociaciones entre las mismas dos clases sin los roles, es dificil saber que significa la asociación.
Con los roles, es claro que una asociación representa la ciudad de la que parte el vuelo y la otra la ciudad de destino.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

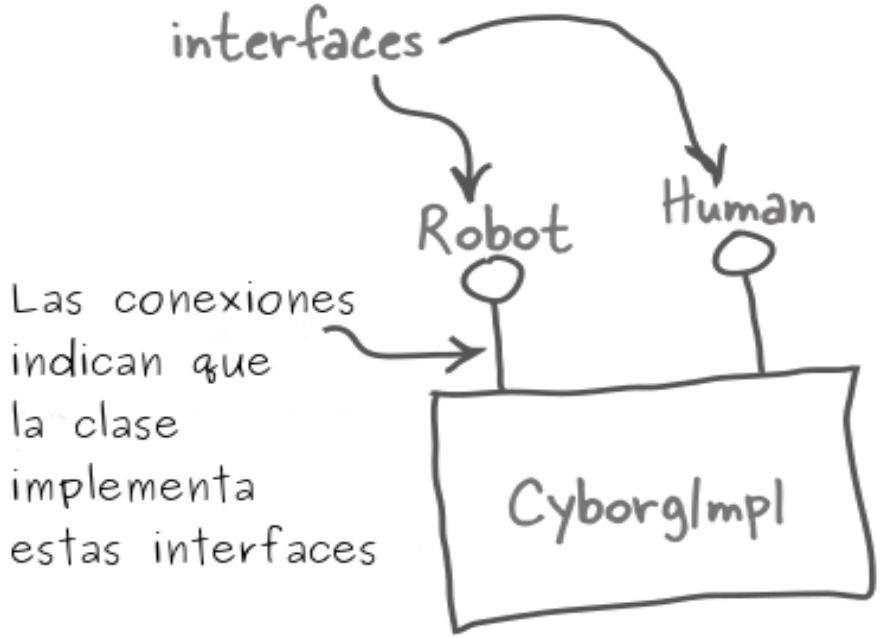
Interfaces

Las Interfaces se dibujan igual que las clases. La única diferencia es el agregado de la palabra “interface” en la parte de arriba (rodeada por los caracteres << >>).
Una clase que implementa una interfaz es conectada a la interfaz vía una linea punteada con un triangulo apuntando a la interfaz.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

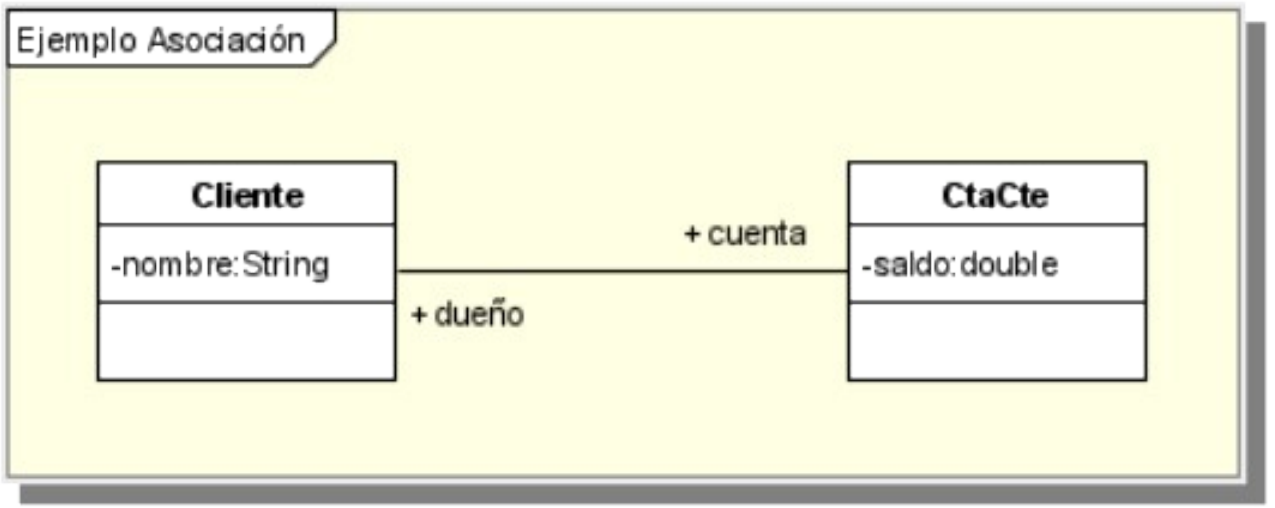
Interfaces

Cuando dibujamos diagramas de clases que implementan muchas interfaces, dibujar cada interface implementada puede volverse inmanejable. UML provee una notación alternativa para indicar la implementacion de interfaces. Se conoce como “lollipop notation” porque la interface es representada con un circulo pequeño unido a una clase por una linea continua. Notar que con esta notacion, no se listan las operaciones definidas por la interfaz.



UML a Código

Asociación Bidireccional con multiplicidad 0..1 o 1

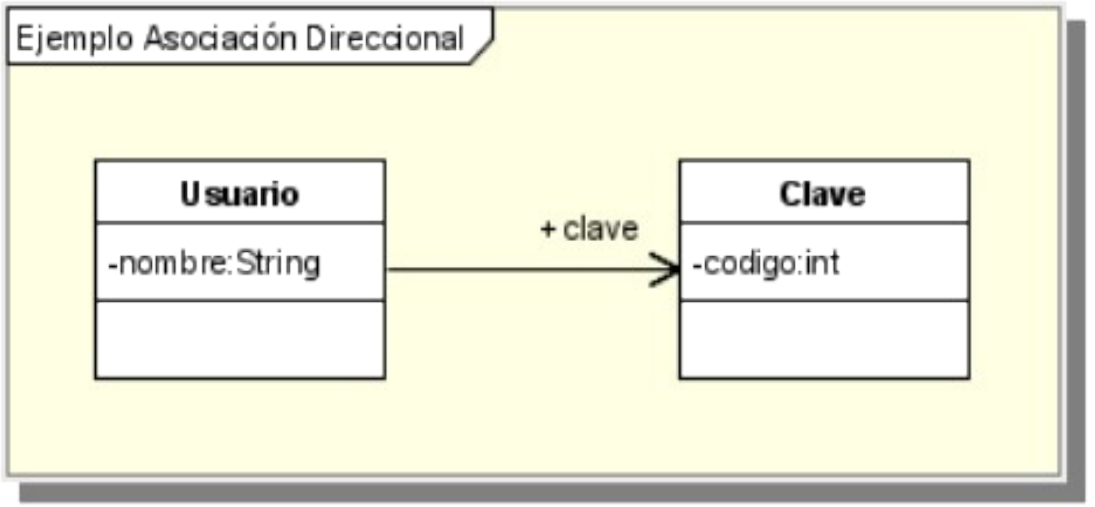



```java
public class Cliente {

    private String nombre;

    public CtaCte cuenta;

}
```

```java
public class CtaCte {

    private double saldo;

    public Cliente dueño;

}
```

UML a Código

Asociación Direccional con multiplicidad 0..1 o 1

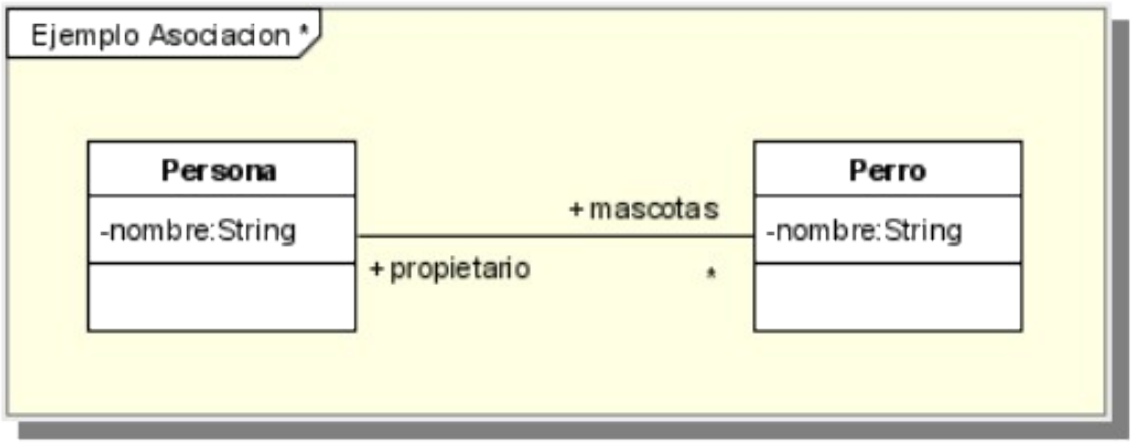



```
public class Usuario {

    private String nombre;

    public Clave clave;

}
```

```
public class Clave {

    private int codigo;

}
```

UML a Código

Asociación Bidireccional con multiplicidad *

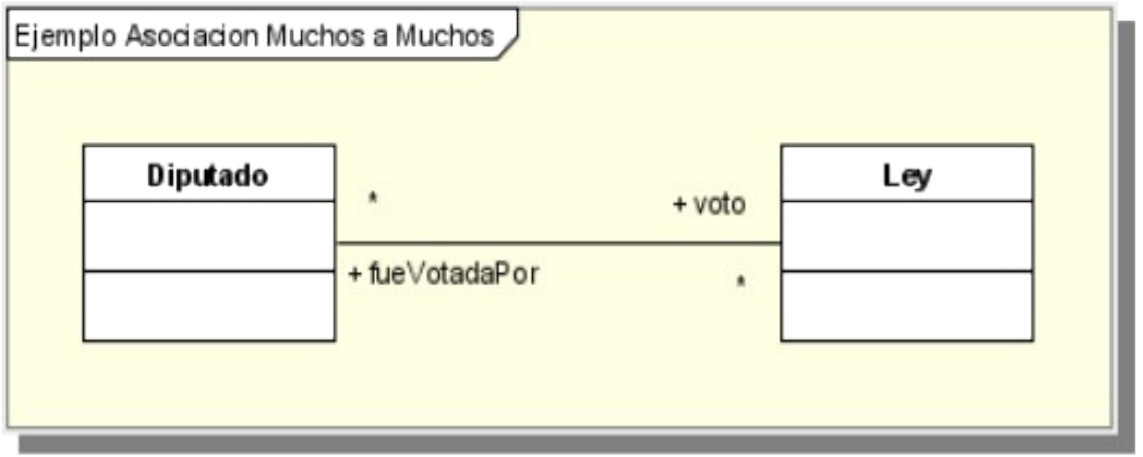



```java
public class Persona {

    private String nombre;

    public java.util.Collection mascotas = new java.util.TreeSet();

}
```

```java
public class Perro {

    private String nombre;

    public Persona propietario;

}
```

UML a Código

Asociación Bidireccional con multiplicidad *

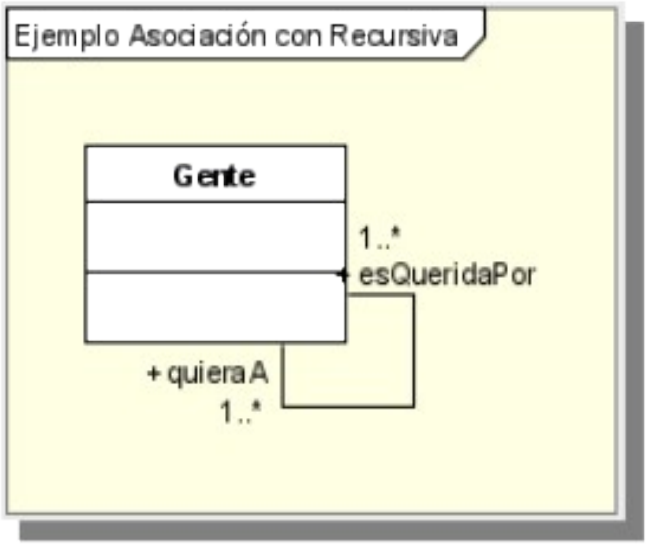



```
public class Diputado {

    public java.util.Collection voto = new java.util.TreeSet();

}
```

```
public class Ley {

    public java.util.Collection fueVotadaPor = new java.util.TreeSet();

}
```

UML a Código

Asociación recursiva

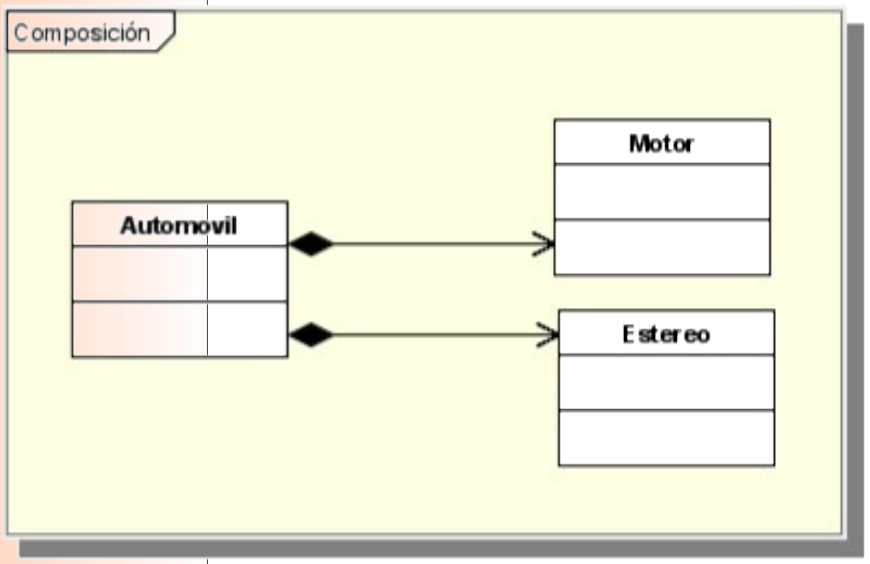



```
public class Gente {

    public java.util.Collection quieraA = new java.util.TreeSet();

    public java.util.Collection esQueridaPor = new java.util.TreeSet();

}
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

UML a Código

Composición

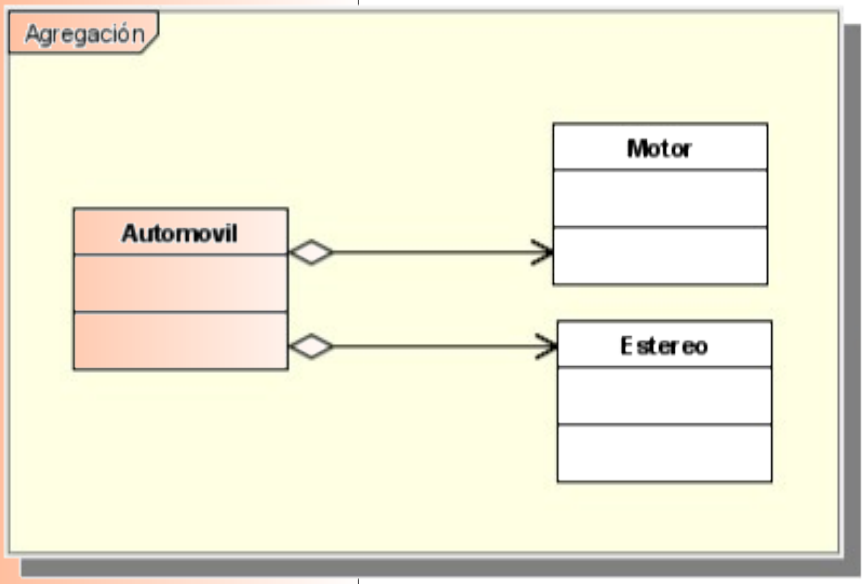



```
public class Automovil {

    public Estereo estereo;
    public Motor motor;

    public Automovil() {
        estereo = new Estereo();
        motor = new Motor();
    }

}
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

UML a Código

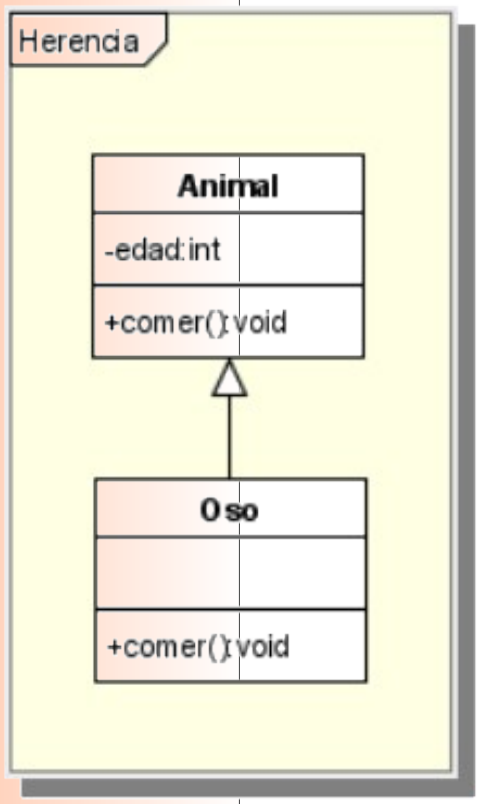
Agregación




```
public class Automovil {

    public Estereo estereo;
    public Motor motor;

    public Automovil() {
    }

    public void ensamblar(Estereo e, Motor m) {
        estereo = e;
        motor = m;
    }

}
```

UML a Código

Herencia



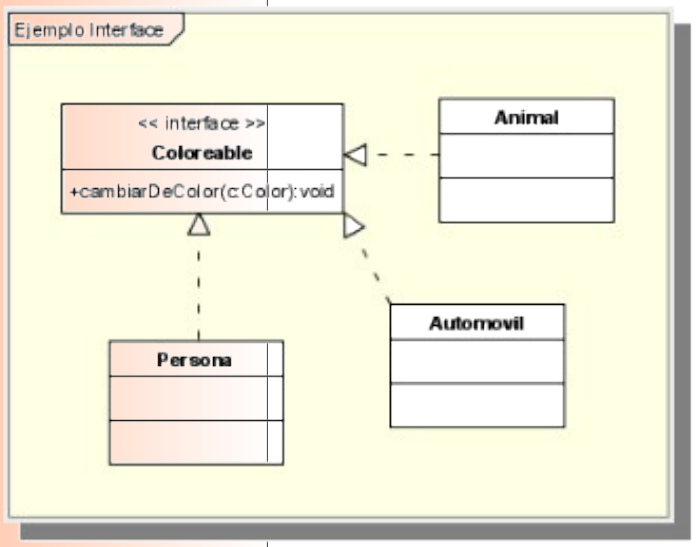

```java
public class Animal {

    private int edad;

    public void comer() {
        // your code here
    }

}

public class Oso extends Animal {

    public void comer() {
        // Para el oso significa otra cosa...
    }

}
```

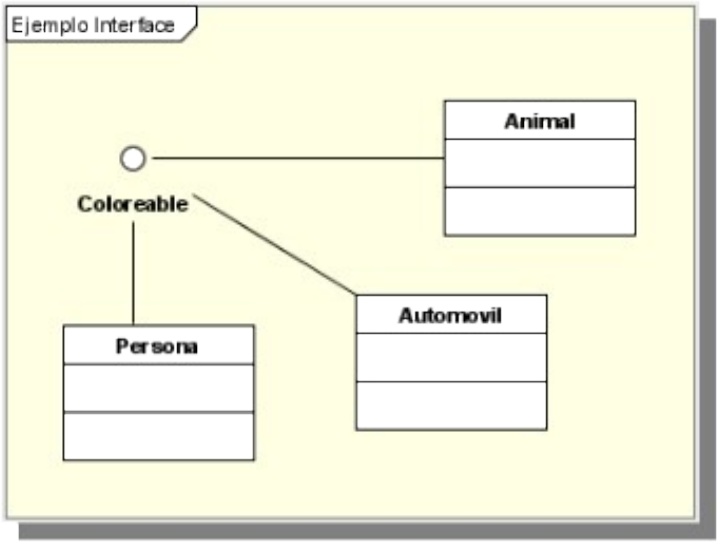
Según el lenguaje, puede ser necesario hacer explícito el override

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

UML a Código

Interfaz






```
public interface Coloreable {

    public void cambiarDeColor(Color c);
}
```

```
public class Persona implements Coloreable {

    public void cambiarDeColor(Color c) {
        // Se debe implementar
    }
}
```

```
public class Automovil implements Coloreable {

    public void cambiarDeColor(Color c) {
        // Se debe implementar
    }
}
```

```
public class Animal implements Coloreable {

    public void cambiarDeColor(Color c) {
        // Se debe implementar
    }
}
```