Unidad 4

# LENGUAJE DE RESTRICCIÓN DE OBJETOS


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# Lenguaje de Restricción de Objetos

## Unidad 4.1

- **OBJETIVOS**

- Definir el lenguaje de restricciones de objetos (OCL) para expresar restricciones y navegaciones sobre un modelo orientado a objetos.

# OCL


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# ¿POR QUÉ UTILIZAR OCL?

- *Un modelo es un conjunto consistente y coherente de elementos de la realidad que tienen características y restricciones*
- *Un modelo gráfico como UML no es suficiente para una especificación no ambigua*
- *Existe la necesidad de establecer restricciones adicionales sobre el modelo*
- *Muchas veces las restricciones se escriben en lenguaje natural*

*¿Qué problemas surgen con este tipo de especificaciones?*

# ¿POR QUÉ UTILIZAR OCL?

- *Una solución: LENGUAJES FORMALES*
  - *Problemas: necesidad de usuarios con una fuerte formación matemática*

- *Una alternativa: OCL*
  - *Lenguaje de características formales*
  - *Fácil de leer*
  - *Fácil de escribir*

# ¿ QUÉ NO ES OCL ?

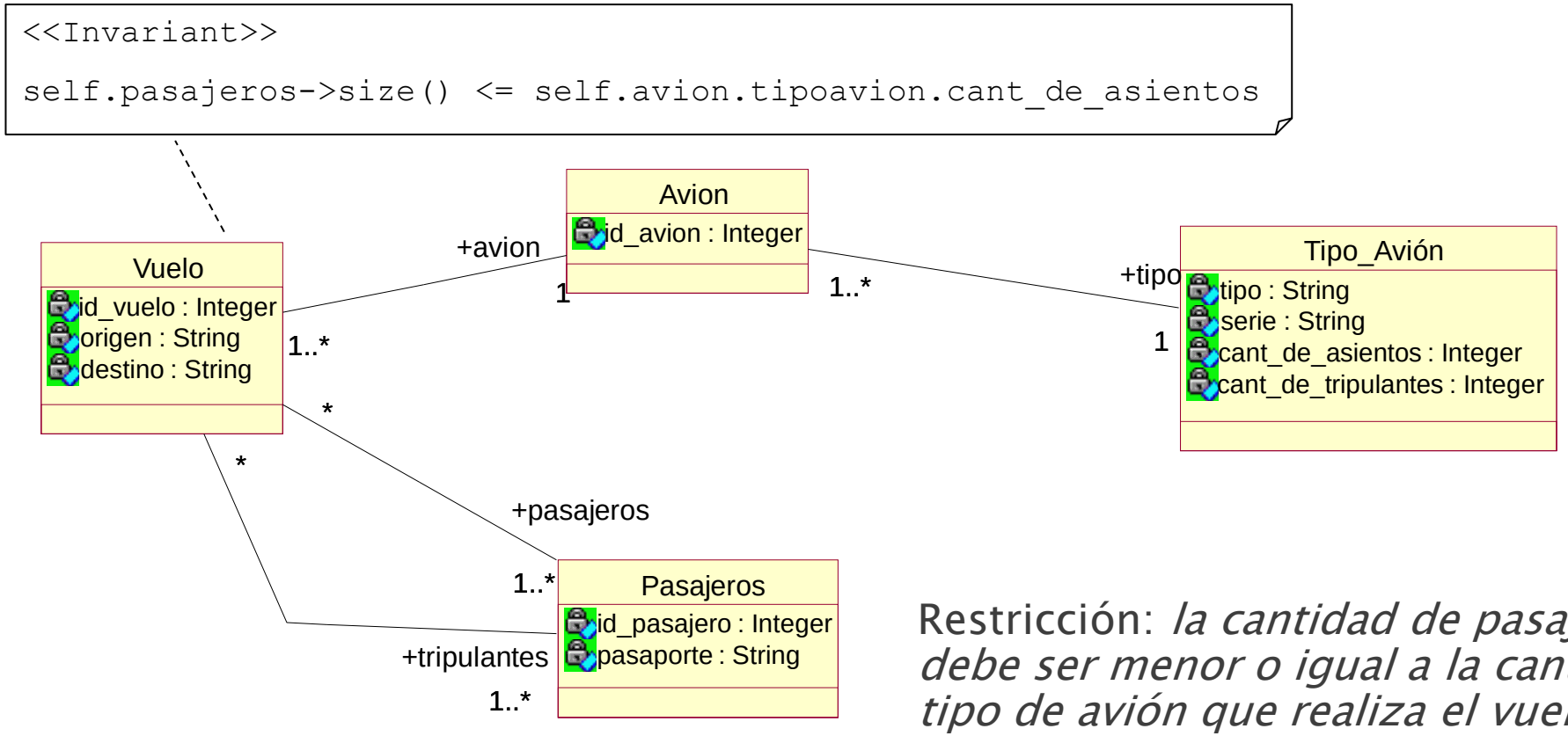
- *NO es un lenguaje de programación*
- *NO es posible escribir lógica de programación*
- *NO es posible invocar operaciones que no sean una consulta*
- *NO considera aspectos de implementación*

# ¿ PARA QUÉ SIRVE OCL ?

*Sirve para:*

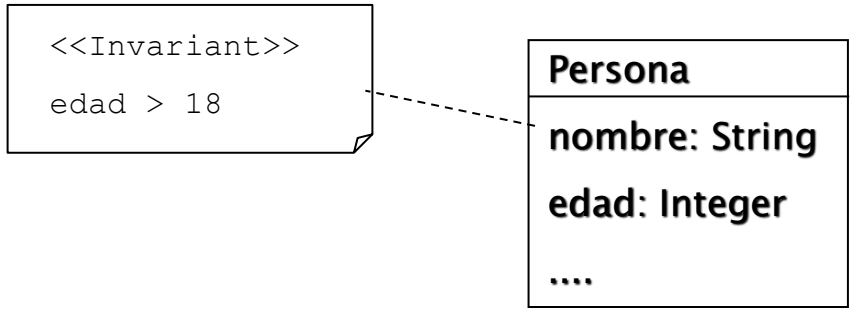
- *Especificar reglas de negocio*
- *Definir la semántica de UML*
- *Especificar PIM para MDA*
- *Especificar modelos precisos y completos a partir de la construcción de modelos UML/OCL combinados*

# UN EJEMPLO




Restricción: *la cantidad de pasajeros de un vuelo debe ser menor o igual a la cantidad de asientos del tipo de avión que realiza el vuelo.*

# CONTEXTO DE UNA EXPRESIÓN

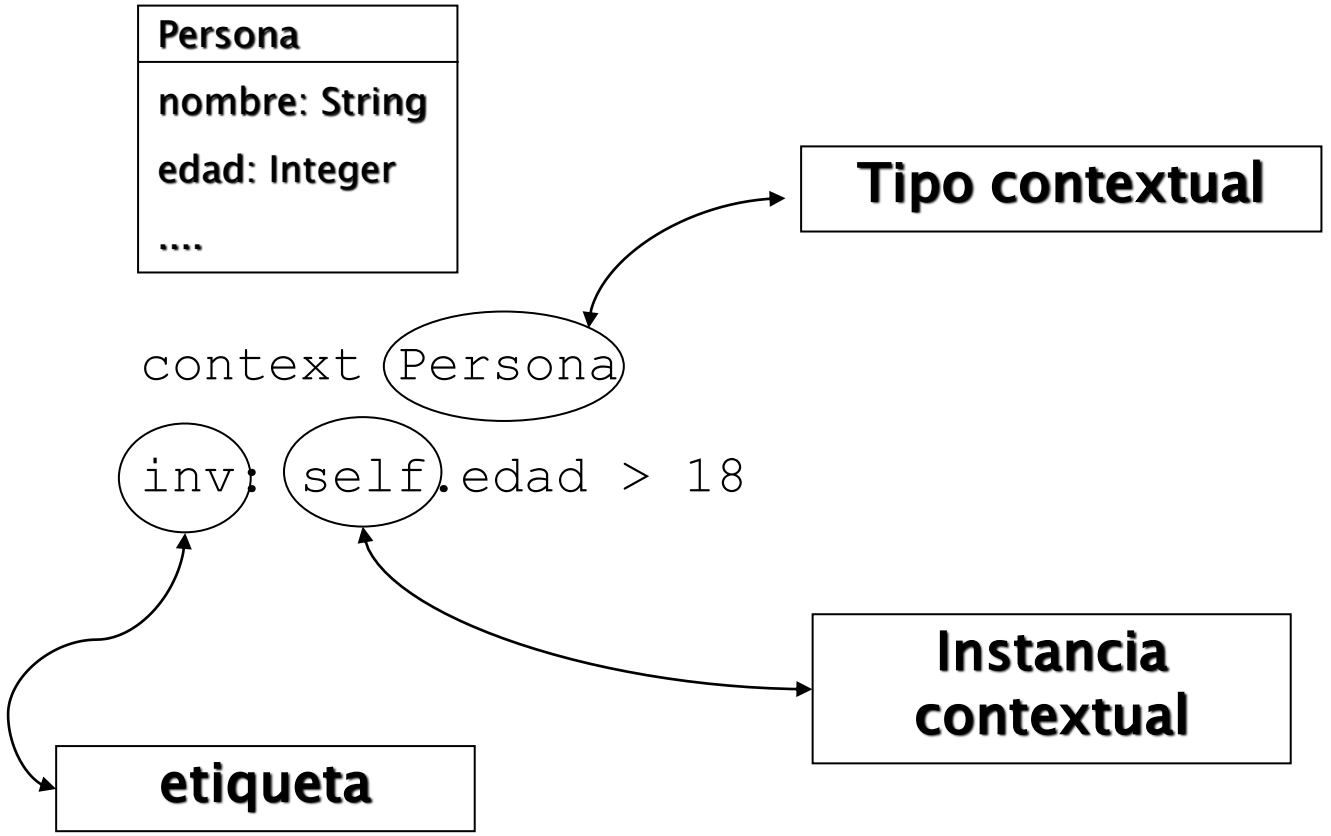



```
context Persona

inv: edad > 18
```

Las expresiones se pueden mostrar en un diagrama UML o no.

El vínculo entre una entidad en un diagrama UML y una expresión OCL se denomina **definición de contexto** de una expresión OCL.

# CONTEXTO DE UNA EXPRESIÓN



# Accediendo a propiedades de objetos


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# PROPIEDADES

- Una propiedad puede ser:
  - Un atributo
  - Un extremo de asociación
  - Una operación/método libre de efectos laterales

  Una operación o método se define como libre de efectos laterales si no modifica el estado del sistema

- **Notación "Punto"**: El valor de una propiedad en un objeto se especifica en una expresión OCL con un punto seguido del nombre de la propiedad

```
self.mEdad >= 17
self.obtenerprecio()
```

# PROPIEDAD :ATRIBUTO

Por ejemplo, la siguiente expresión OCL: “la edad de cliente es mayor o igual a 17 años”:

```
context Cliente

inv: self.mEdad >= 17
```

El valor de la subexpresión self.mEdad es el valor de mEdad en la instancia particular de Cliente identificada por self.

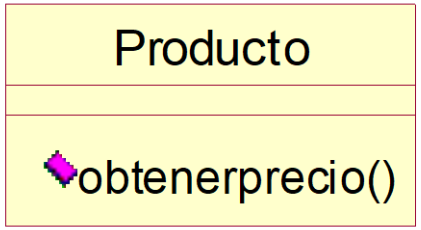
El tipo de la subexpresión self.mEdad es el tipo del atributo mEdad, que es un tipo Integer estándar.

# PROPIEDAD : OPERACIONES

Para referirnos a una operación, también se utiliza la notación punto

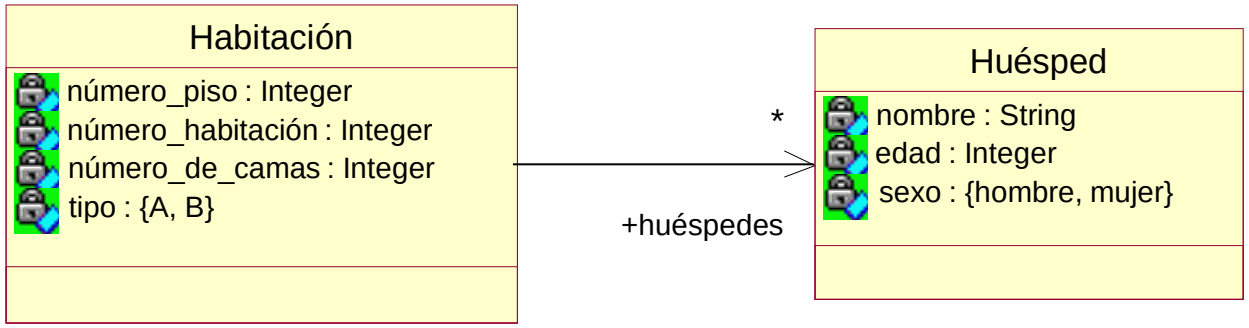
Por ejemplo:

```
context Producto

Inv: self.obtenerprecio() > 0
```



# NAVEGACIONES

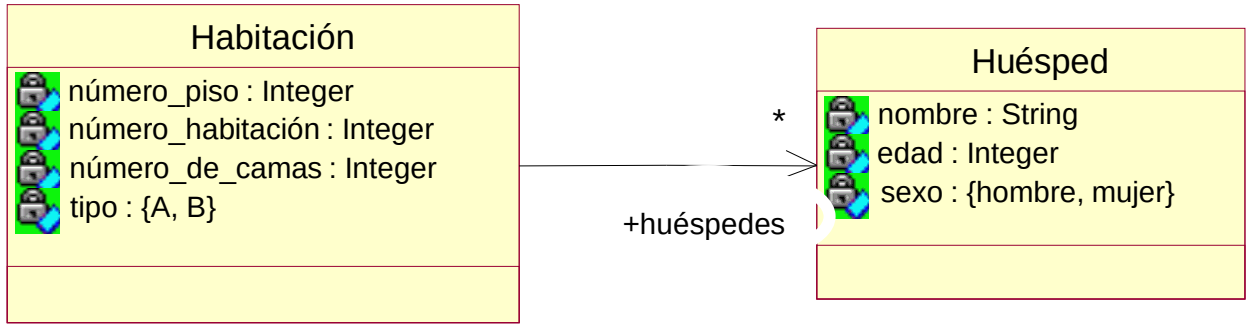
- Las navegaciones nos permiten hacer referencia a objetos que están asociados con la instancia contextual.
- Hay dos tipos de navegaciones: simples y combinadas.



# NAVEGACIONES

Requerimiento: la cantidad de huéspedes de una habitación no debe superar la cantidad de camas de la habitación.

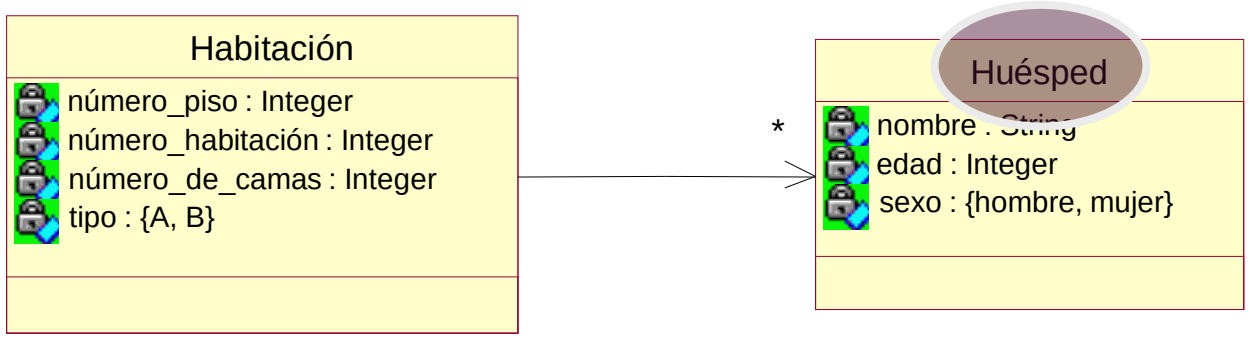
```
context Habitación

inv: self.huespedes->size() <= self.número_de_camas
```




Se utiliza el nombre de rol (del extremo opuesto de una relación) que vincula la clase donde se define la expresión con otra clase del diagrama.

# NAVEGACIONES

Si se omite el nombre de rol, se usa como tal el nombre de la clase




```
context Habitación
inv: self.Huesped->size() <= self.número_de_camas
```

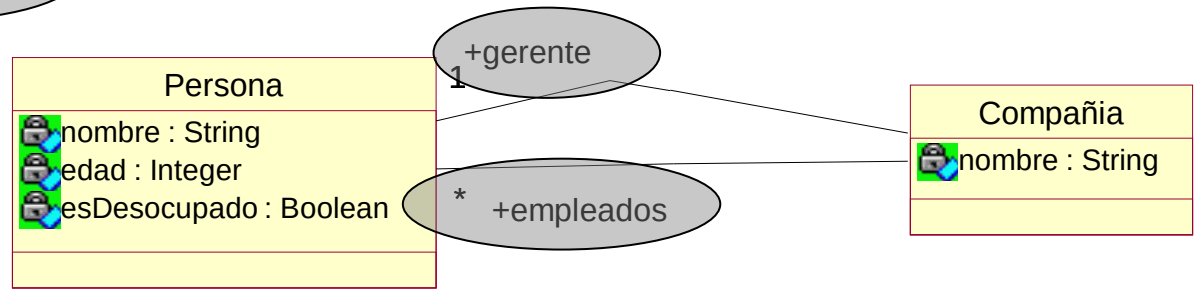
# PROPIEDAD: EXTREMOS FINALES DE ASOCIACIONES

A partir de un objeto específico, es posible navegar una asociación en el diagrama de clase para referirnos a otros objetos y sus propiedades.

objeto.nombredelextremoFinal

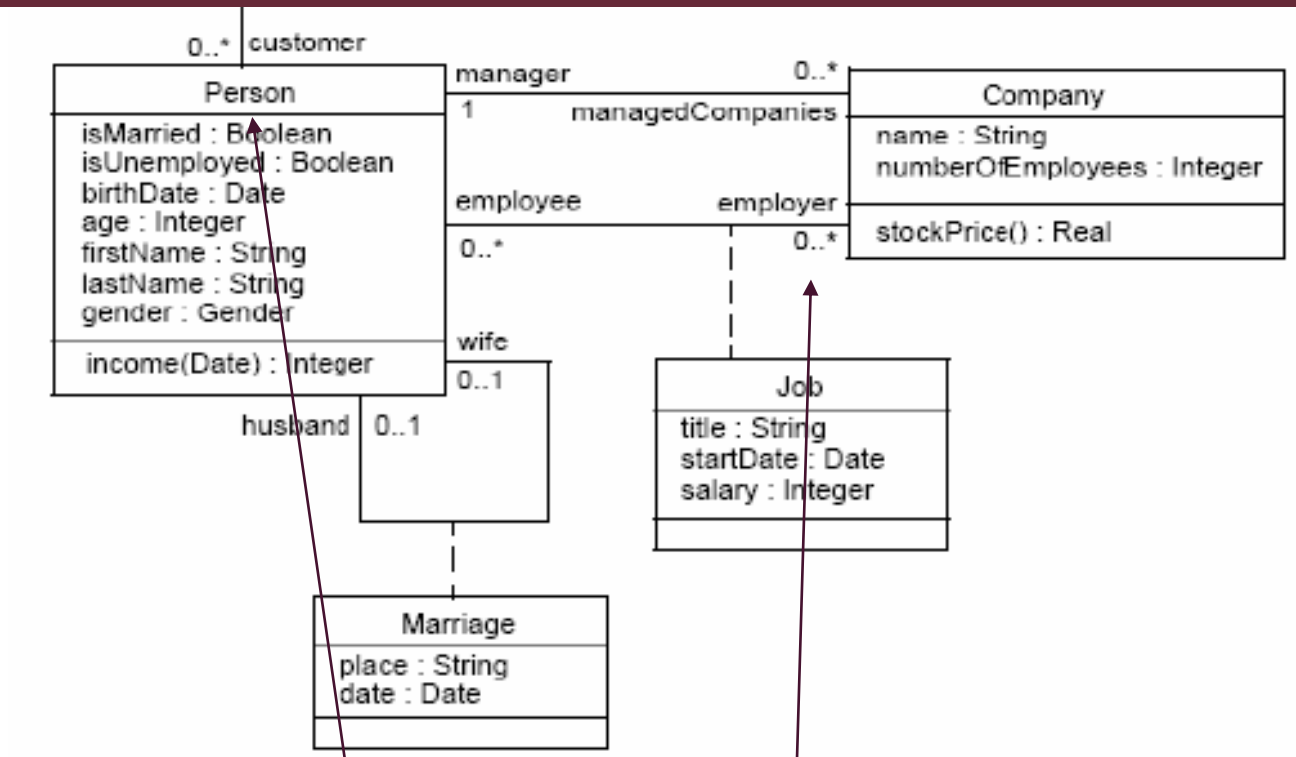
El valor de la subexpresión es un objeto o conjuntos de objetos, dependiendo de la multiplicidad del extremo final de la asociación.

```
context Compañia

inv: self.gerente.esDesocupado = false

inv: self.empleados->notEmpty()
```




En self.empleados->notEmpty() estamos accediendo a la propiedad notEmpty en el conjunto self.empleados

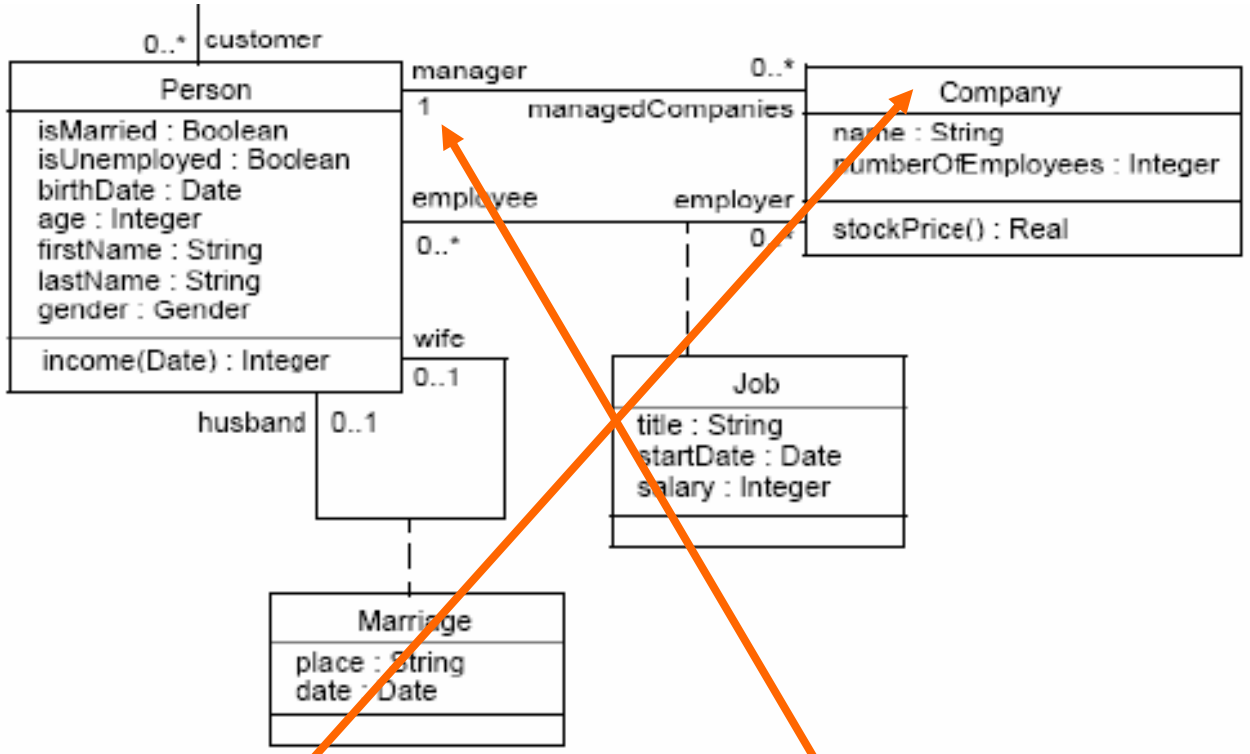
# MULTIPLICIDAD




*Si la multiplicidad del extremo es *, es una colección*

```
Context Person  inv:
self.employer ...  (una  colección)
```

# MULTIPLICIDAD




*Si la multiplicidad del extremo es 0 ó 1, es un objeto*

```
Context Company inv: Self.manager... (Un objeto)
```

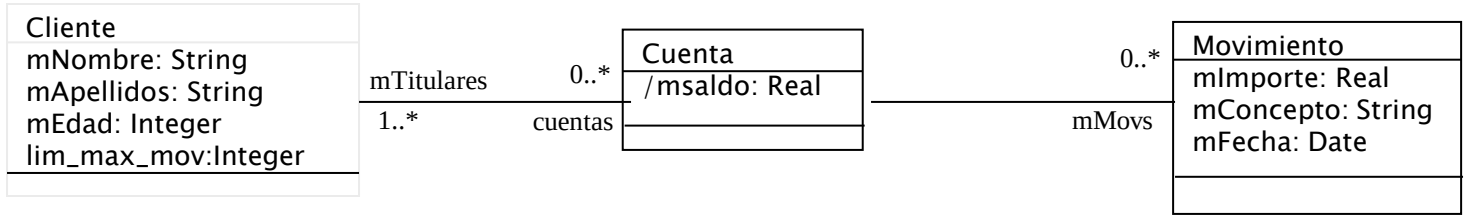
# COMBINANDO PROPIEDADES

Las propiedades se pueden combinar para escribir expresiones más complejas.

Una regla importante es que una expresión OCL siempre se evalúa a un objeto específico de un tipo especifico. Luego de obtener el resultado, es posible aplicar otra propiedad para obtener un nuevo resultado.

Cada expresión OCL puede ser leída y evaluada de izquierda a derecha

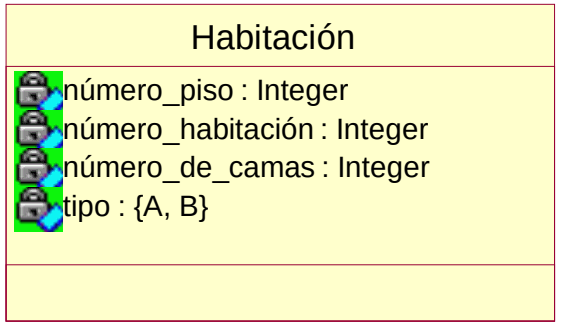
```
context Cliente

inv: self.lim_max_mov >= self.cuentas.mMovs->size()
```



# INVARIANTES

- Un invariante es una condición que debe ser verdadera para todas las instancias de un tipo específico en cualquier momento.
- Su tipo contextual es un clasificador.

```
context Habitación

inv: self.número_de_camas <= 10

context Habitación

inv: número_de_camas <= 10

context h: Habitación

inv: h.número_de_camas <= 10
```



# PRE Y POST CONDICIONES

- Una precondición /postcondición es una condición que debe ser verdadera antes /después de la ejecución de una operación.
- La instancia contextual self es una instancia del tipo que es dueño de la operación.
- La declaración incluye la palabra context, seguida del contexto y de la declaración de la operación y el tipo.
- Las etiquetas pre y post declaran si se trata de una pre/postcondición.

```
context Empleado::aumentarsalario(
cantidad: Real ): Real

pre: salario > 0

post: self.salario = self.salario@pre +
cantidad and

      result = self.salario
```

# PRE Y POST CONDICIONES

En una postcondición, nos podemos referir a los valores de cada propiedad de un objeto en dos momentos en el tiempo:

- El valor de la propiedad al comienzo de la operación o método (utilizamos el postfijo @pre)
- El valor de la propiedad una vez que la operación o método se ha ejecutado.

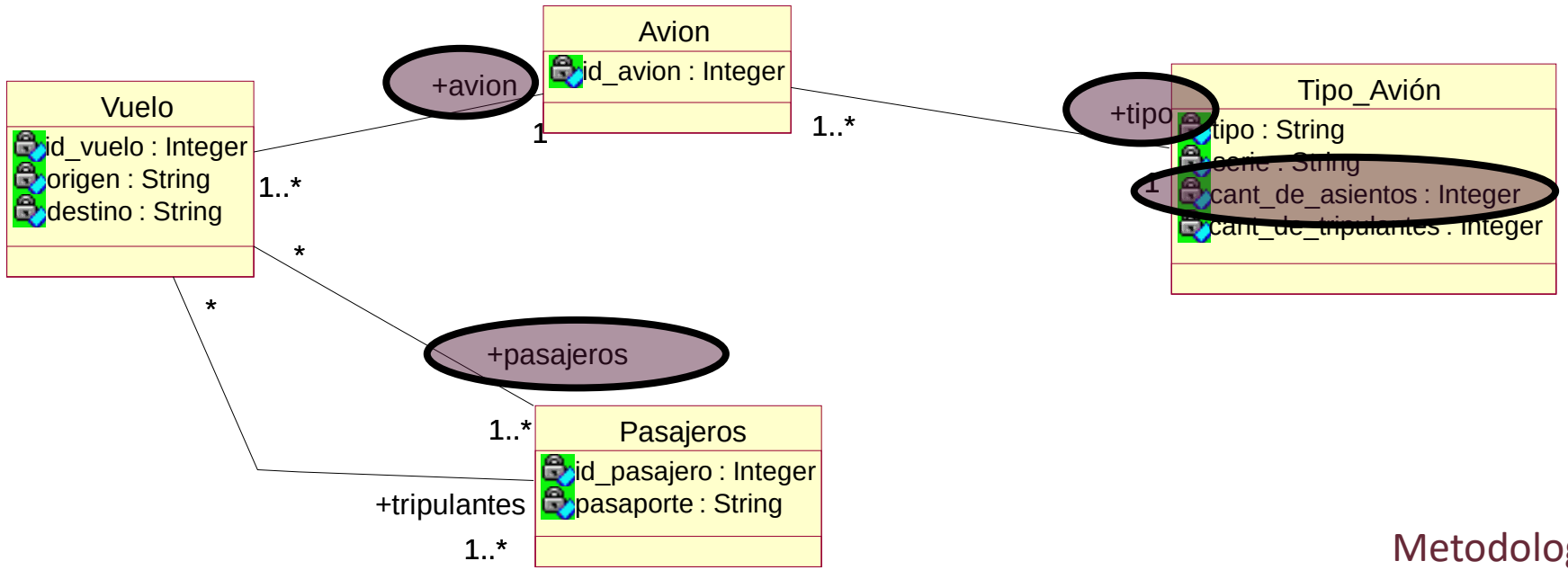
**Nota: '@pre' solo se permite en expresiones del tipo postcondicion !**

```
context Empleado::aumentarsalario( cantidad: Real ):
Real

pre: salario > 0

post: self.salario = self.salario@pre + cantidad and

        result = self.salario
```

# NAVEGACIONES COMBINADAS

Las navegaciones no se limitan a una única asociación. Las expresiones OCL pueden estar encadenadas navegando un conjunto de asociaciones

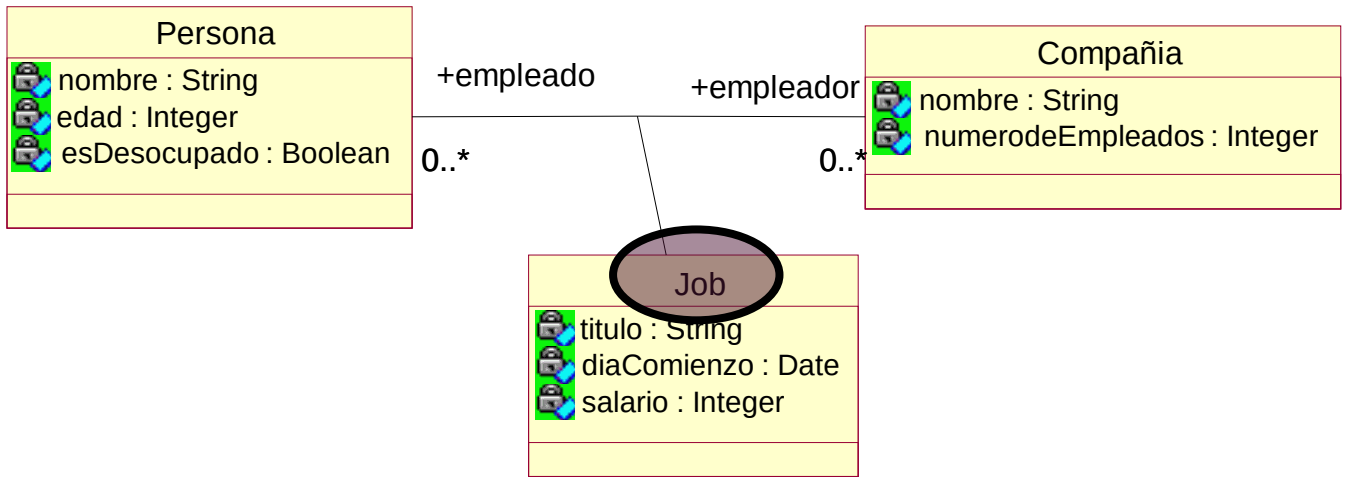
```
context Vuelo

inv: self.avion.tipo.cant_de_asientos >= self.pasajeros->size()
```



# NAVEGACIONES A CLASES DE ASOCIACION

Para especificar la navegación a clases de asociación, OCL utiliza un punto y el nombre de la clase de asociación

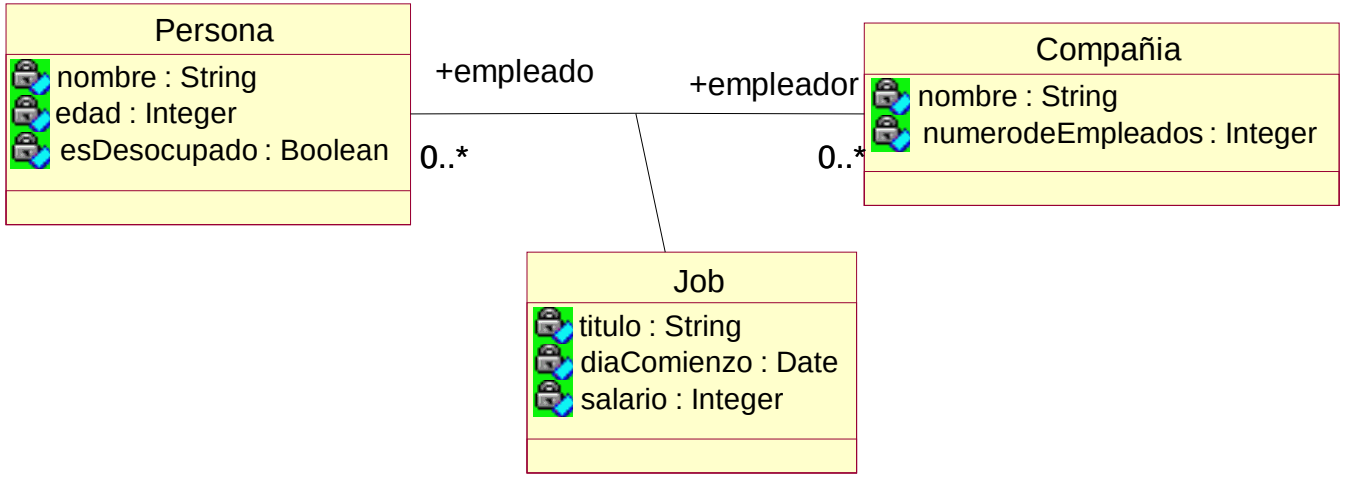
```
context Persona

inv: self.job ...
```



# NAVEGACIONES DESDE CLASES DE ASOCIACIÓN

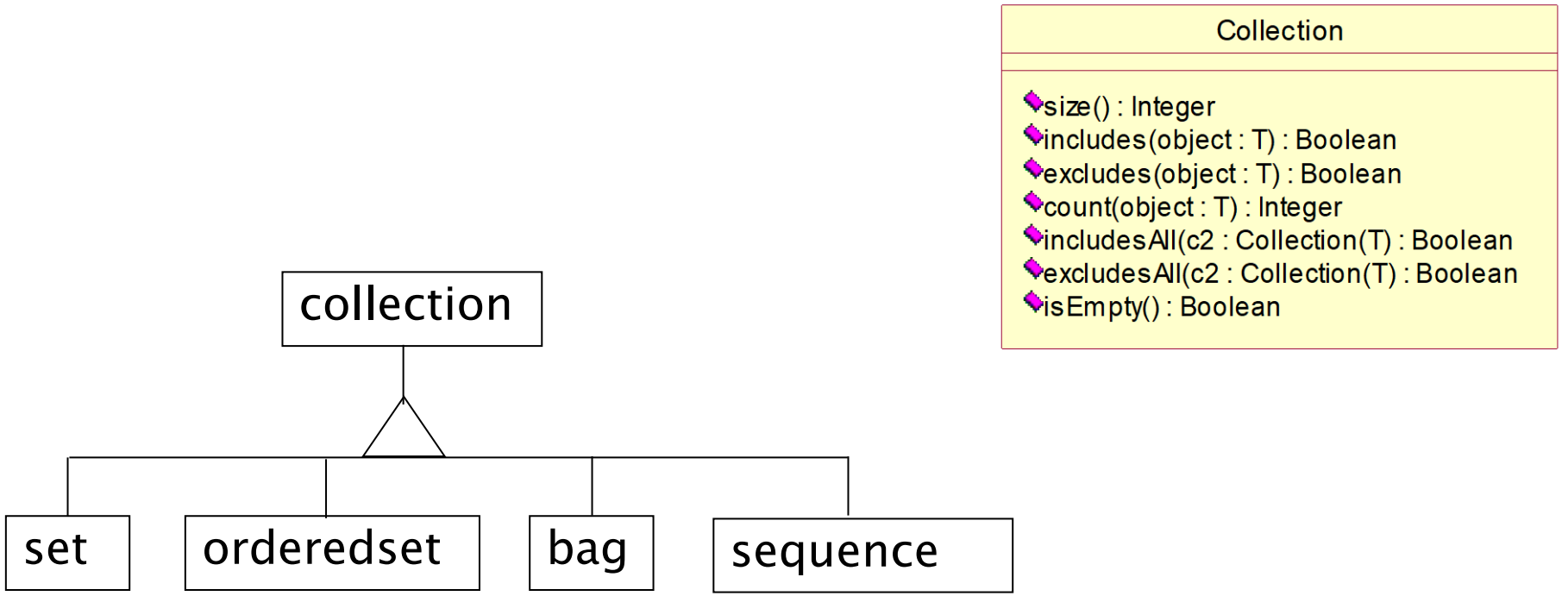
Es posible navegar desde la clase de asociación a los objetos que participan en la asociación.

```
context Job

inv: self.empleado.edad > 21
```



# EL METAMODELO DE OCL

En el metamodelo de OCL se define la jerarquía de colecciones de OCL y las operaciones de colección.



# COLECCIONES

- El tipo **Collection** es un tipo abstracto, con tipos de colección concretos como sus subtipos.

  - **Set (Conjunto)**
  - **OrderedSet (Conjunto ordenado)**
  - **Bag (Bolsa)**
  - **Sequences (Secuencias)**

# NAVEGACIONES Y COLECCIONES

Los tipos de colecciones juegan un importante rol en las expresiones OCL

- Una única navegación resulta en un conjunto (Set),
- Navegaciones combinadas en un Bag,
- Navegaciones sobre asociaciones adornadas con {ordered} resultan en un OrderedSet.

# Tipos básicos.


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# TIPOS BÁSICOS

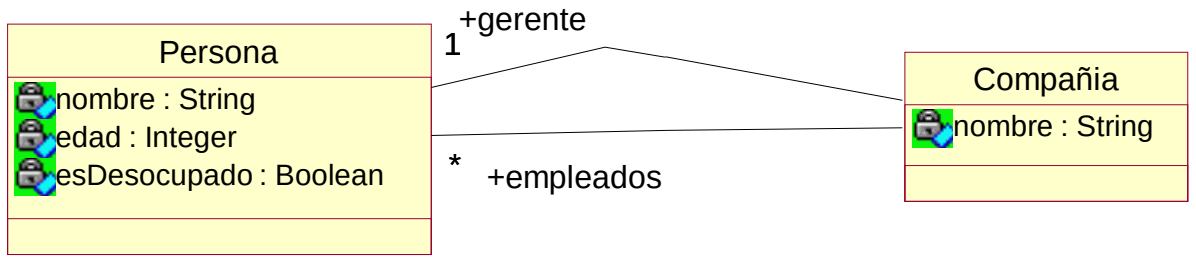
- En OCL, se definen un número de tipos básicos, los cuales están disponibles para el modelador en todo momento. Estos tipos son valores predefinidos y son independientes de cualquier modelo de objetos.
- Tipos Básicos:
  - Boolean: true, false
  - Integer :1, -5, 2, 34, 26524, ...
  - Real: 1.5, 3.14, ...
  - String: 'esto es un string'
  - Colecciones (definidas anteriormente)
- Operaciones definidas en los tipos
  - Integer: *, +, -, /, abs(), etc.
  - Real: *, +, -, /, etc.
  - Boolean: and, or, xor, not, implies, if-then-else-endif
  - String: concat(), size(), substring()

# NAVEGACIONES QUE RESULTAN EN UN SET

Las navegaciones simples resultan en un Set

El valor de una navegacion simple es un objeto o conjuntos de objetos, dependiendo de la multiplicidad del extremo final de la asociación.

```
context Compañia

        inv: self.gerente.esDesocupado = false

        inv: self.empleados->notEmpty()
```



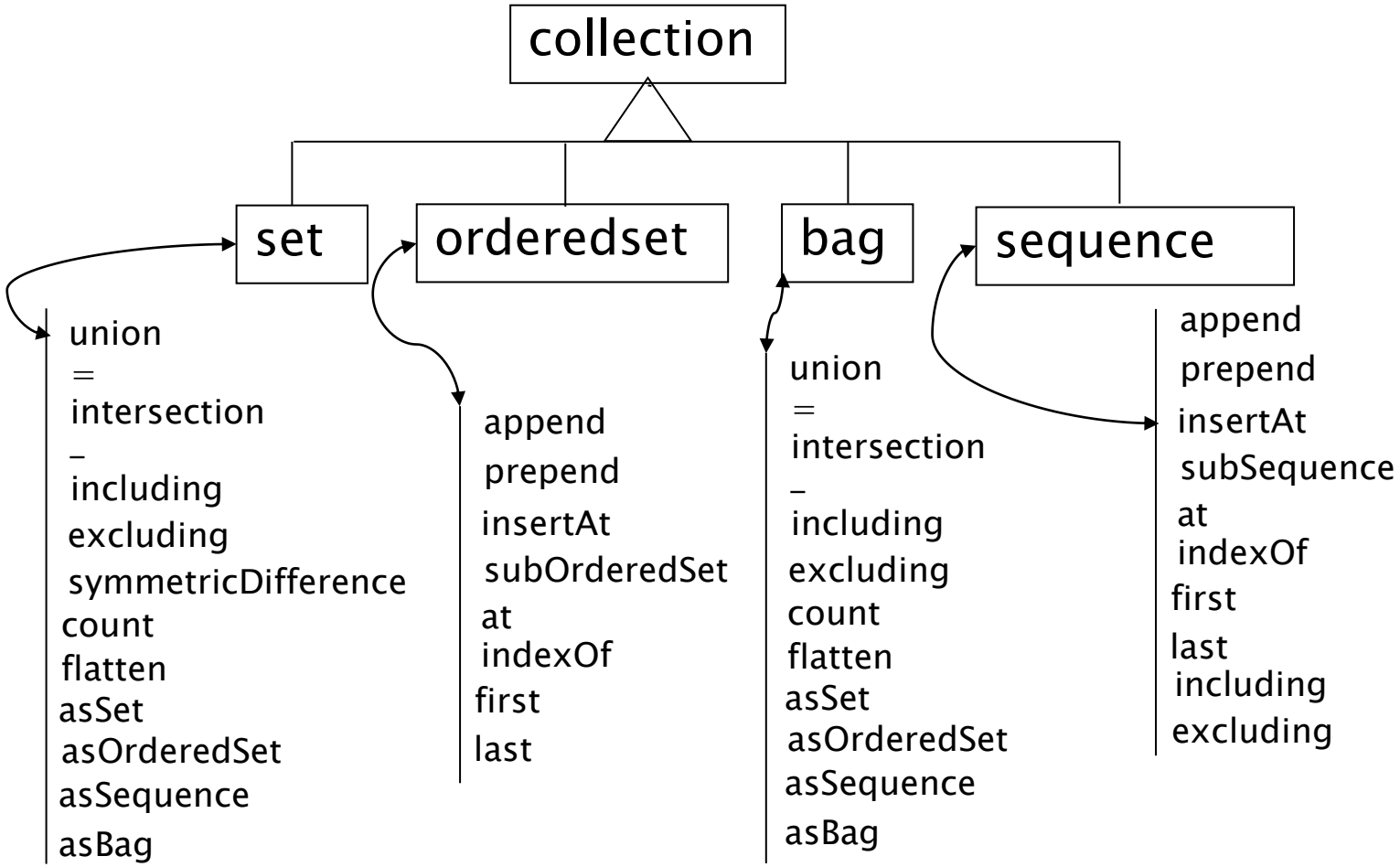

```
.... self.gerente->size() = 1
```

# Colecciones y operaciones de colección.


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# OPERACIONES DE COLECCIÓN



# OPERACIONES DE COLECCIÓN

OCL define muchas operaciones en tipos de colección. Estas operaciones permiten contar con una manera flexible y poderosa de proyectar nuevas colecciones desde colecciones existentes.

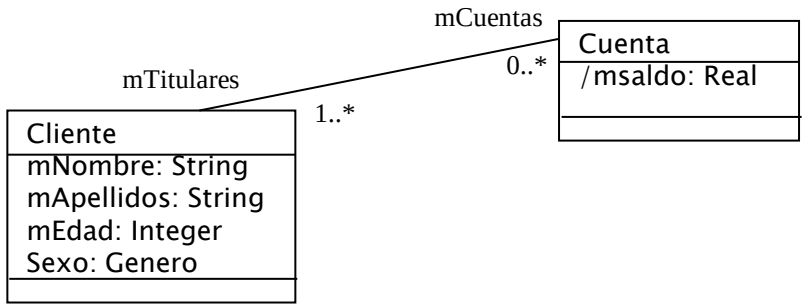
**Notación flecha "->":**

```
colección->operaciónDeColeccion()
```

# OPERACIÓN SIZE ()

La operación predefinida size() nos permite obtener la cantidad de elementos de una colección.

```
context Cliente

inv: self.mCuentas->size() < 10
```



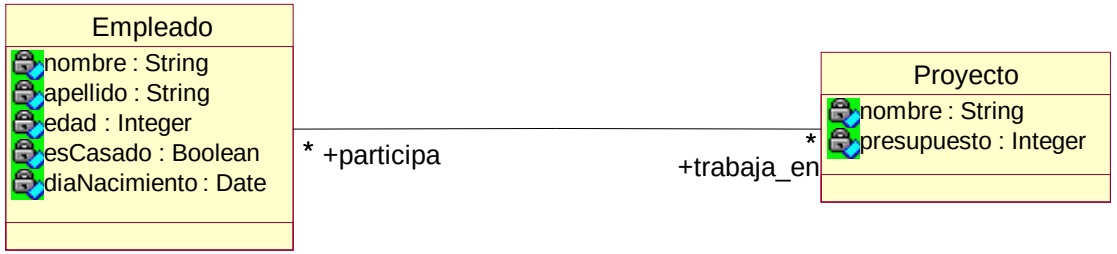
# OPERACIÓN SELECT

- Permite obtener un subconjunto específico de una colección.
- **select** es una operación sobre una colección y es especificada utilizando la sintaxis de flecha:

  **collection->select( expresión booleana)**

- El resultado es una colección que contiene todos los elementos de la colección origen para los cuales es verdadera la expresión booleana.
- Para encontrar **el resultado de esta operación**, se evalúa la expresión para cada elemento de la colección. Si el resultado de la evaluación es verdadero para un elemento, este se incluye en la colección resultante.

# OPERACIÓN SELECT




El contexto de la expresión en el argumento select es el del elemento de la colección en el cual el select es invocado.

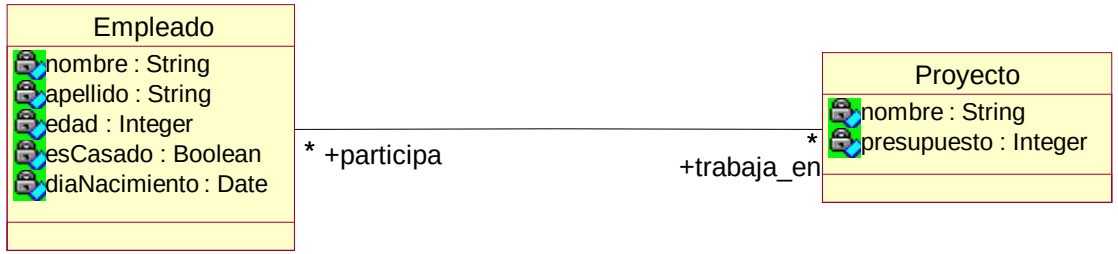
```
context Proyecto

inv: self.participa -> select (edad > 50) ->notEmpty()

context Proyecto

inv: self.participa-> select (e: Empleado | e.edad > 50) ->notEmpty()

context Proyecto

inv: self.participa-> select (e  | e.edad > 50) ->notEmpty()
```

# OPERADOR COLLECT

El operador collect se utiliza cuando queremos derivar una nueva colección a partir de otra, pero que contiene objetos diferentes de la colección original.

Por ejemplo: deseamos obtener una colección de las fechas de cumpleaños de los empleados de la compañía:

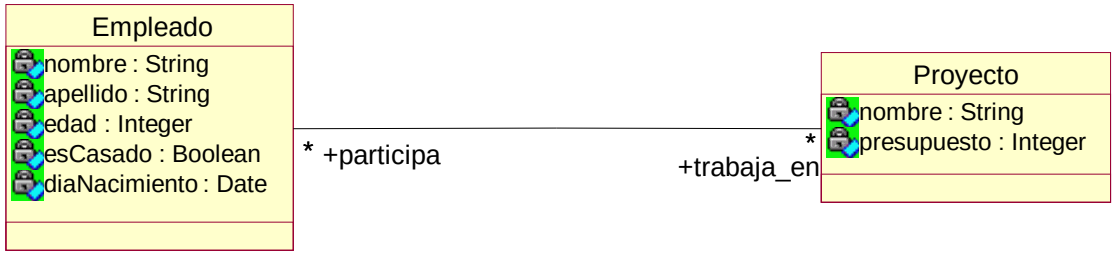
```
self.empleados->collect( fechaNacimiento )

self.empleados->collect( emp : Empleado | emp.fechaNacimiento )
```



# OPERADOR FORALL

Este operador permite especificar una expresión booleana que debe ser verdadera para todos los elementos de una colección.
La exp. forAll tiene como resultado un valor booleano.

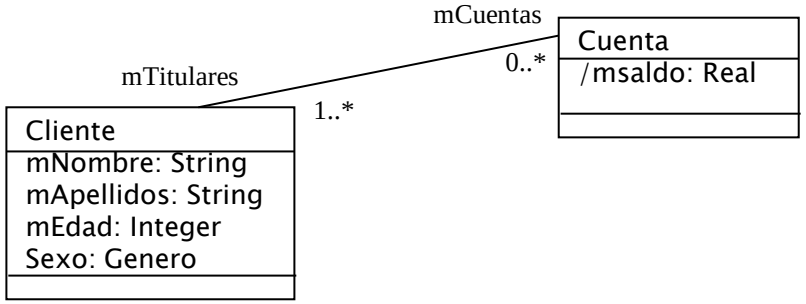
```
context Proyecto
inv: self.participa->forAll( p: Empleado |  p.edad <= 65)
```



# OPERACIÓN EXISTS

Muchas veces es importante saber si al menos para un elemento de una colección se verifica una condición:

```
context Proyecto

inv: self.participa->exists( p | p.nombre = 'Jack')
```

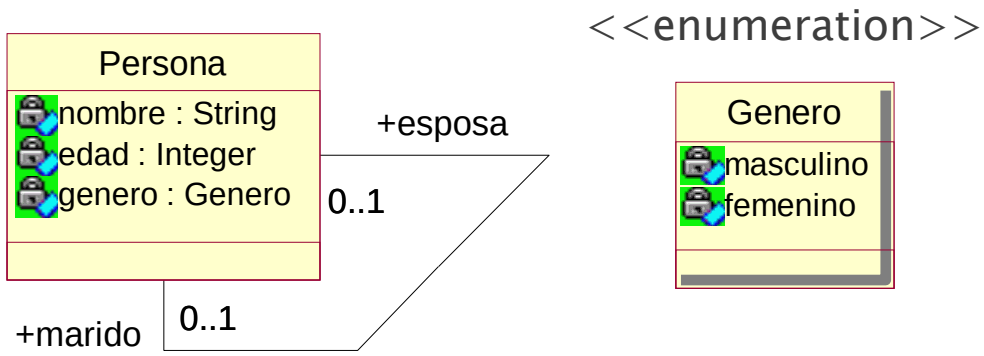

```
context Cuenta

inv: self.mTitulares->exists(c:Cliente|
c.mEdad>=18)
```

# NOTEMPTY ()

En el caso de una asociación con multiplicidad 0..1 es útil verificar si existe un objeto o no cuando navegamos la asociación

context Persona

```
inv: self.esposa->notEmpty() implies self.esposa.genero = Genero::femenino
```



# OPERACIÓN COUNT()

- self-> count(object)
  - El número de veces que la colección (self) incluye el objeto "object"

Bag{1, 2, 3, 2, 4, 2}->count(2) = 3

  - Diferencia entre count() y size()

# OPERADORES LÓGICOS /I

- Operador **not**

| b | not b |
|---|---|
| false | true |
| True | false |

- Operador **and**

| b1 | b2 | B1 and b2 |
|---|---|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | True |

(bi and b2) = (b2 and b1)

# OPERADORES LÓGICOS /2

- Operador **or**

(bi or b2) = (b2 orb1)

| b1 | b2 | b1 or b2 |
|---|---|---|
| false | false | false |
| false | true | true |
| True | false | true |
| True | true | true |

- Operador **xor**

(bi xor b2) = (b2 xorb1)

| b1 | b2 | b1 xor b2 |
|---|---|---|
| false | false | false |
| false | true | true |
| True | false | true |
| true | true | false |

# OPERADORES LÓGICOS /3

- Operador **implies**

  b1 **implies** b2

Si b1 es true, entonces b2 debe ser true (si b1 es false, nada puede decirse de b2)

```
context Person inv:
self.wife->notEmpty implies self.wife.age>=18 and self.husband->notEmpty implies
self.husband.age>=18
```

- Expresión **if**

  **if** b **then** e1 **else** e2 **endif**

b es una expresión booleana y e1 y e2 son expresiones OCL

```
if (count <= 100) then x/2 else 0 endif
```

# SUGERENCIAS

- Utilice OCL cuando quiera modelar cosas que los gráficos no permiten.
- Recuerde que OCL no es un lenguaje de programación.
- Si quiere ser más estricto en la sintaxis, puede usar sentencias OCL en la descripción de pre y post condiciones en los casos de uso.
- Realice navegaciones en OCL en el diagrama de clases. Le servirá para entender mas las multiplicidades en las asociaciones.

# AUTOEVALUACIÓN / I

Comprendí los conceptos más importantes de la unidad 4.1 si puedo definir y dar ejemplos de:

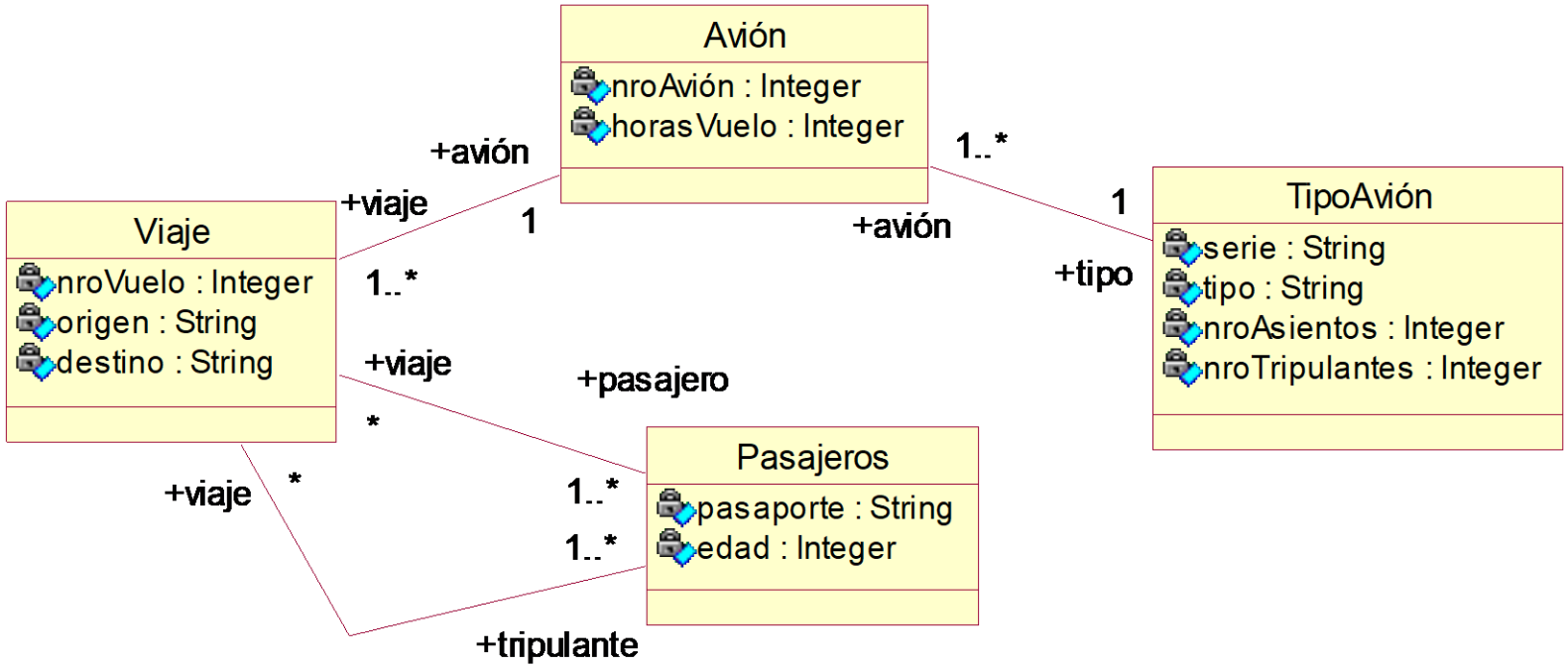
- Sentencia OCL
- Invariantes
- Propiedades
- Navegación
- Colecciones
- Operaciones en la colecciones

# AUTOEVALUACIÓN /2

Comprendí los conceptos más importantes de la unidad 3.1, si :

- Entiendo cómo puedo complementar los modelos UML con sentencias OCL.
- Comprendo cuál es la ventaja de usar OCL en lugar de un lenguaje informal.
- Entiendo qué significa que OCL es un lenguaje declarativo.
- Comprendo qué es un invariante en una clase.
- Entiendo cómo se específica una propiedad (atributo, operación, nombre de rol) en un diagrama de clases
- Comprendo cómo navegar una asociación a partir de los nombres de roles.
- Entiendo cuándo en la navegación hago referencia a un conjunto o un objeto a partir de la multiplicidad del extremo opuesto de la asociación.
- Comprendo la diferencia entre los distintos tipos de colecciones.

# GUÍA DE APRENDIZAJE



# ESTABLECER LAS SIGUIENTES RESTRICCIONES UTILIZANDO OCL

- Para cada tipo de avión, el número de asientos para tripulantes no puede ser mayor al número de asientos para pasajeros
- Los viajes no pueden tener el mismo destino que el origen
- La cantidad de horas de vuelo de un avión debe ser menor a 1000
- Para aviones del tipo “Delta” de la serie “A”, la cantidad de asientos para los pasajeros debe ser de 40
- Los pasajeros de los viajes deben ser mayores de 3 años y menores de 95
- Un viaje debe tener más de 5 pasajeros
- La cantidad de tripulantes de un viaje debe ser menor o igual a la cantidad de pasajeros
- La cantidad de pasajeros de un viaje debe ser menor o igual a la capacidad de asientos del avión asignado al vuelo
- Un avión no puede tener más de 500 viajes asignados en su vida útil si es del tipo “Delta”
- Para los viajes que tengan asignados aviones con una cantidad de horas de vuelo superior a 500 hs, la edad de los tripulantes deben ser mayores a 50 años.

Fin de la clase

UAI
Universidad Abierta Interamericana