

# OCL

**Object Constraint Languaje**

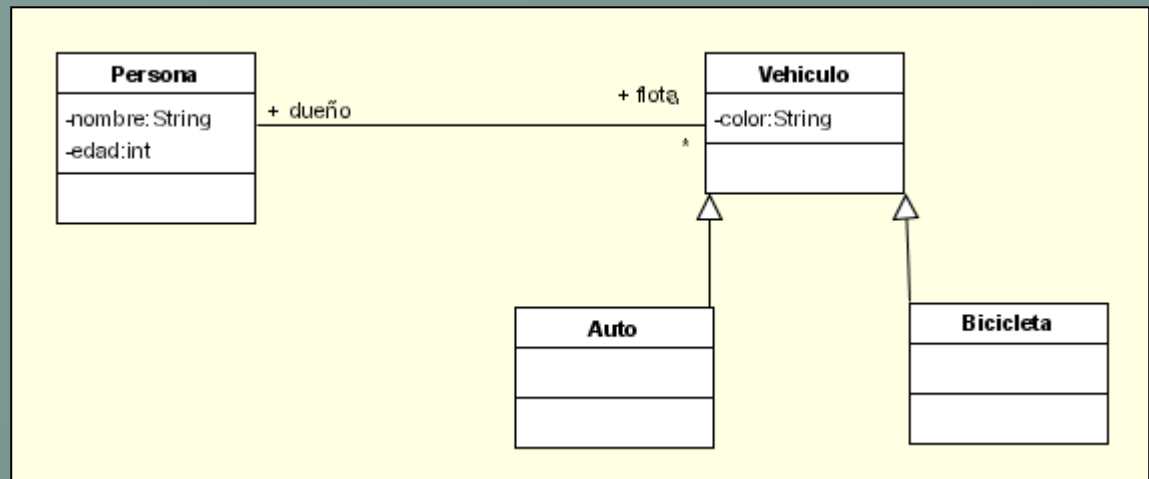
# Object Constraint Language

- Es un lenguaje formal de especificación.
- Define una sintaxis, precisa una semántica.
- Es parte del estándar UML.
- Es relativamente fácil de leer.
- Permite verificar modelos.

# Object Constraint Language

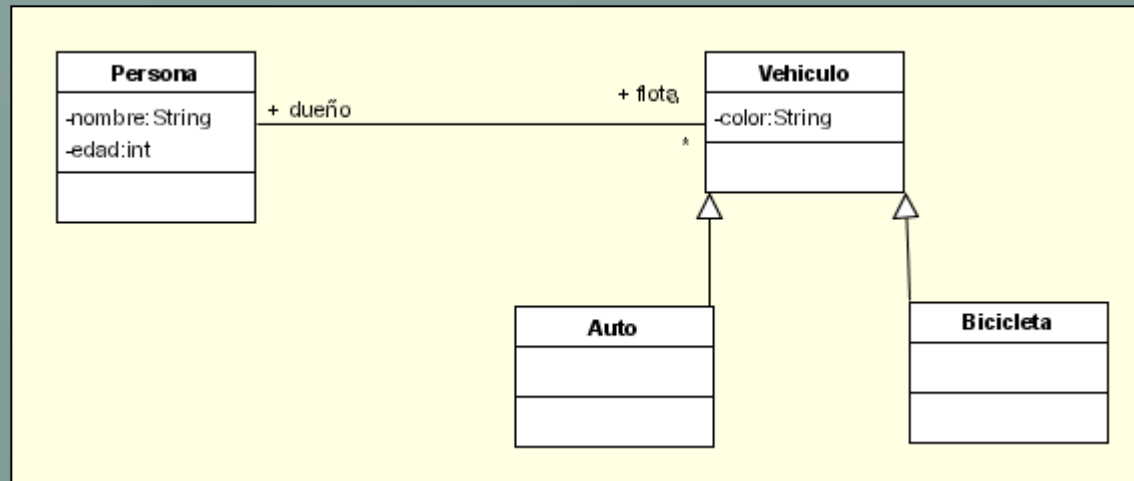
¿Cómo modelar los siguientes requerimientos?

- Posible número de dueños que un vehículo puede tener.
- Edad requerida para los dueños de autos.
- Toda persona deba tener al menos un auto negro.



# Object Constraint Language

“El dueño de un vehículo debe tener al menos 18 años”

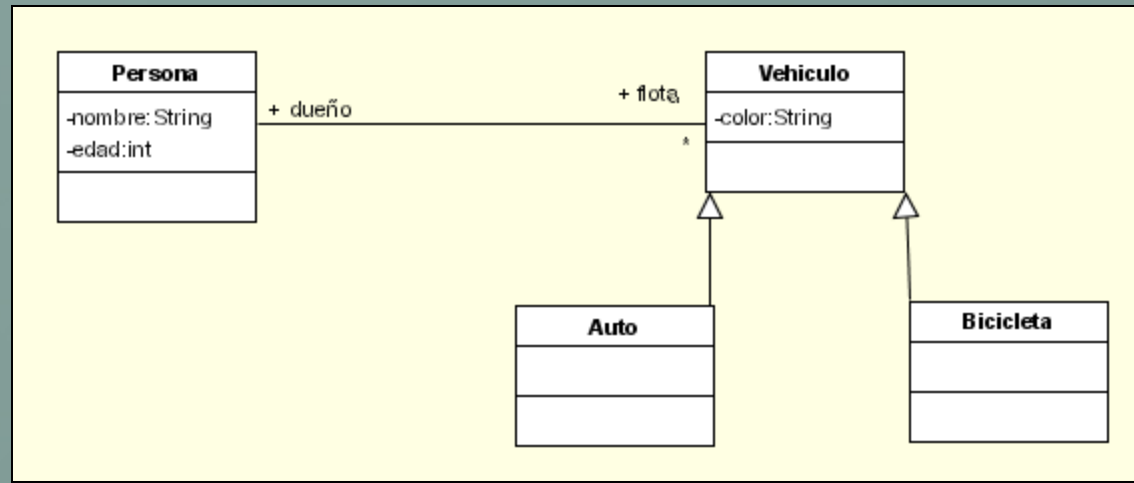


context Vehículo

inv: self.dueño.edad >= 18

# Object Constraint Language

¿Qué significa esto?

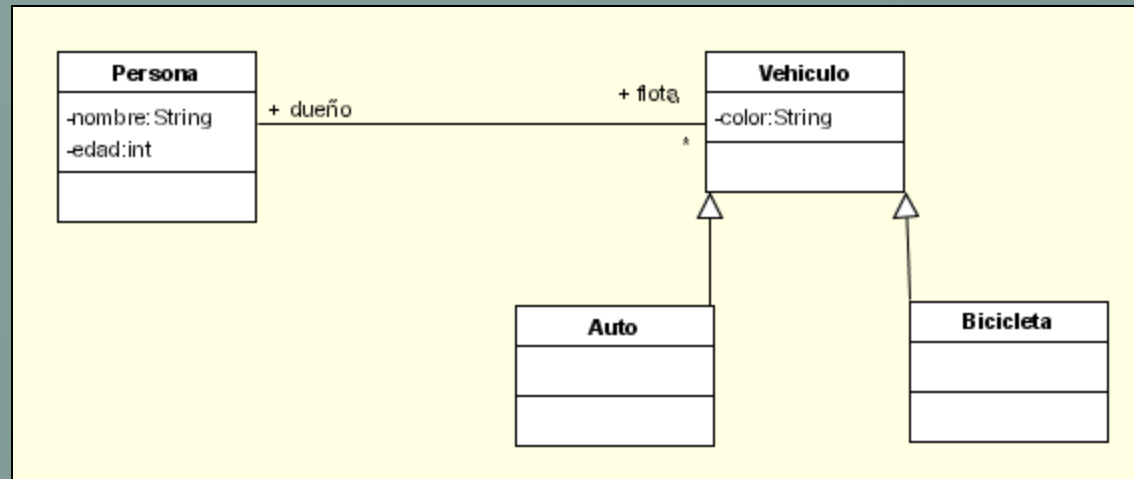


context Persona

inv: self.edad >= 18

# Object Constraint Language

“El dueño de un auto debe tener al menos 18 años”

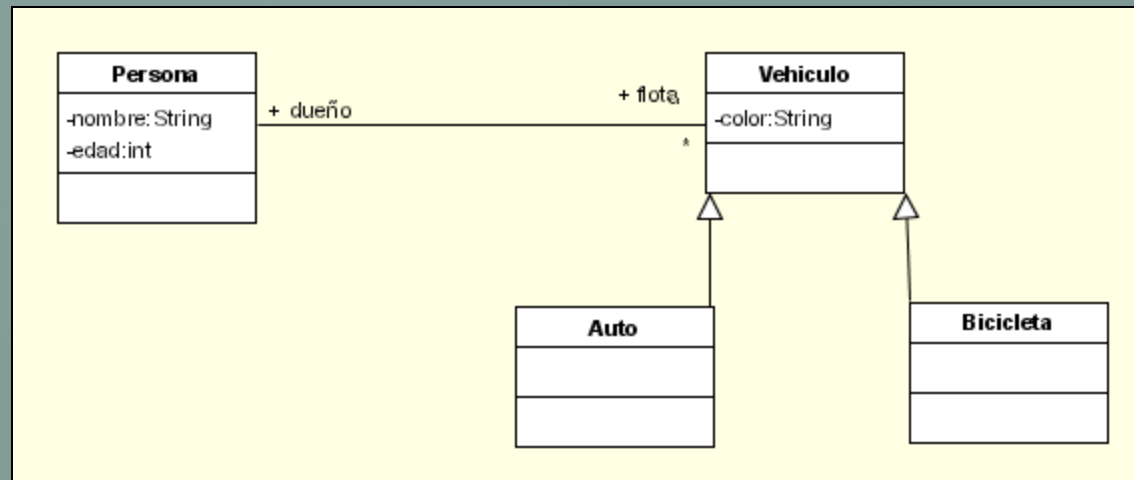


context Auto

inv: self.dueño.edad  $\geq$  18

# Object Constraint Language

“Nadie tiene más de 3 autos”

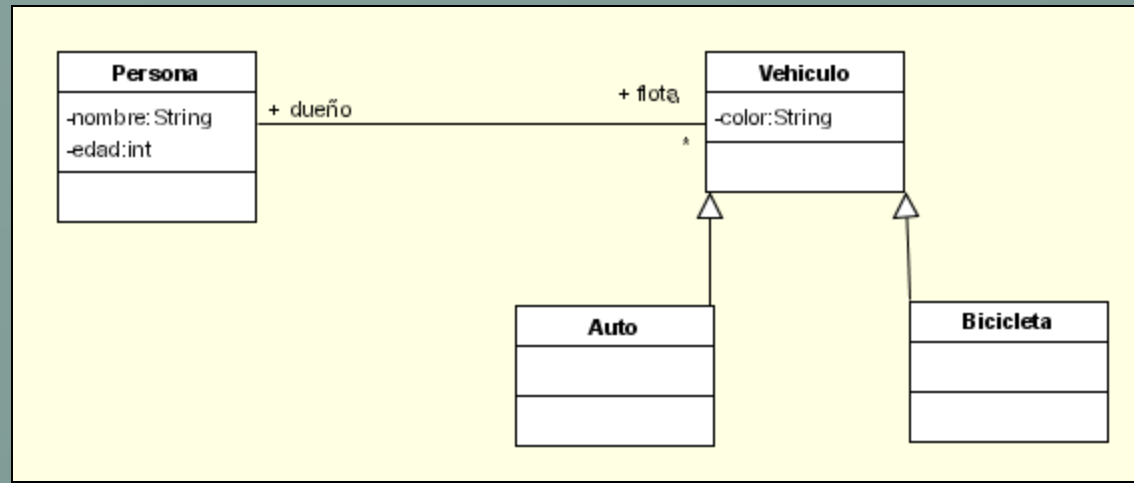


context Persona

inv: self.flota->size <= 3

# Object Constraint Language

“Todos los autos de una persona son negros”



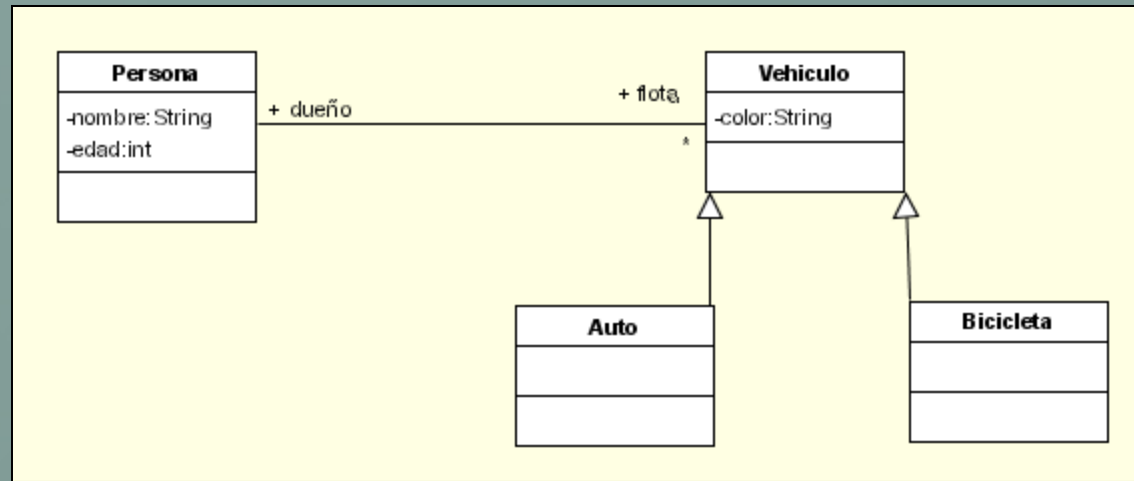
context Persona

inv: self.flota->forAll(v | v.color="negro")



# Object Constraint Language

“Nadie tiene más de 3 vehículos negros”

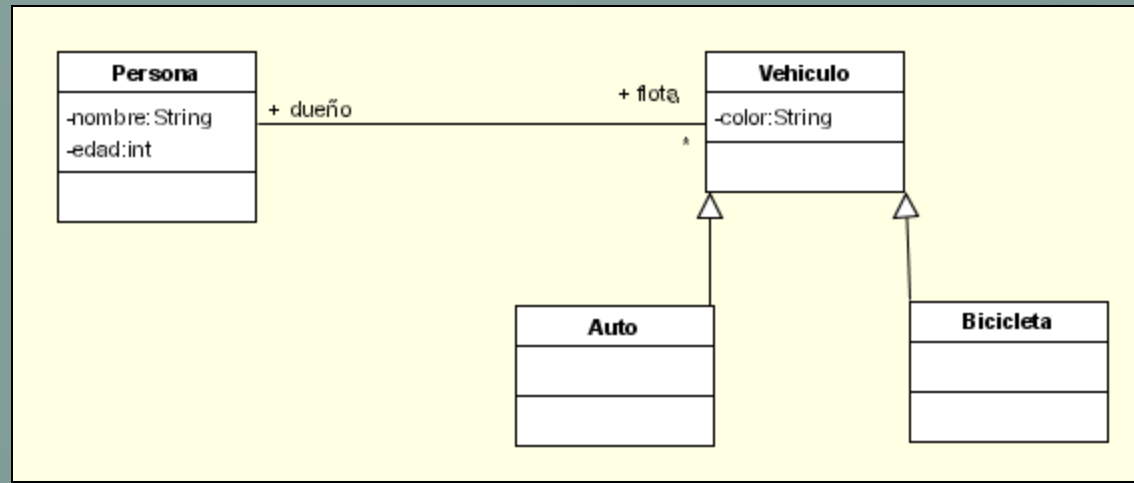


context Persona

inv: self.flota->select(v | v.color="negro")->size <= 3

# Object Constraint Language

“Ninguna persona menor a 18 años, posee un auto”

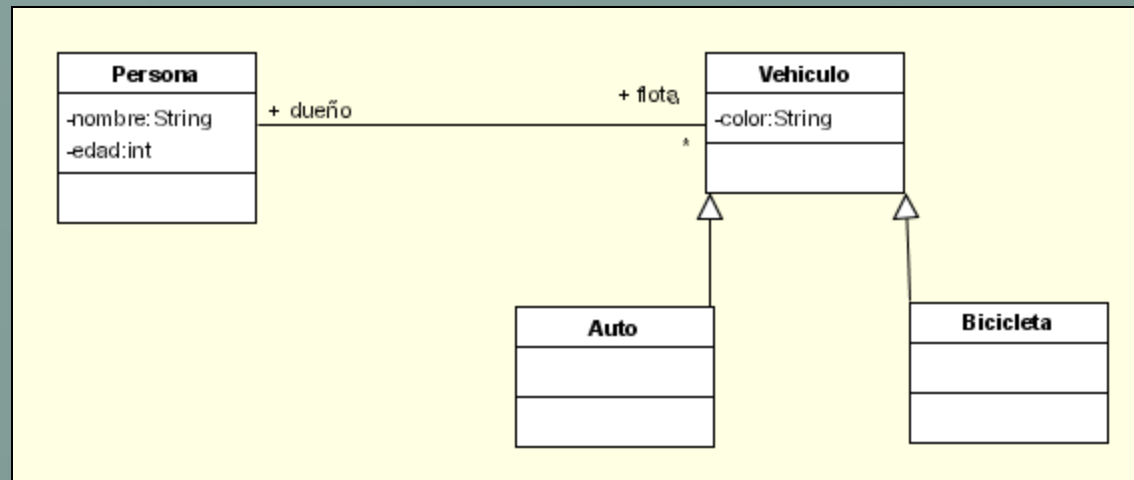


context Persona

inv: self.edad < 18 implies self.flota -> forAll(v | not v.ocllsKindOf(Auto))

# Object Constraint Language

“Existe un auto rojo”

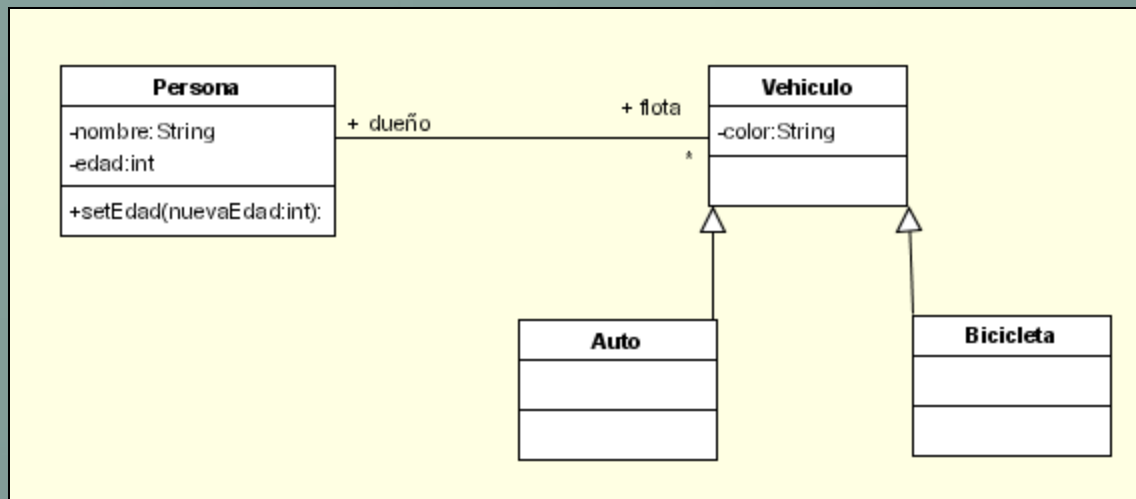


context Auto

inv: Auto.allInstances()->exists(c | c.color="rojo")

# Object Constraint Language

Puede utilizarse para definir precondiciones y postcondiciones de operaciones



```
context Persona::setEdad(nuevaEdad:int)
pre: nuevaEdad >= 0
post: self.edad = nuevaEdad
```

# Object Constraint Language: Resumen

- OCL es utilizado para especificar invariantes de los objetos y pre y post condiciones de las operaciones.
- Hace a los diagramas de clases más precisos.
- Las expresiones OCL utilizan vocabulario del diagrama de clases UML.

# Object Constraint Language: Resumen

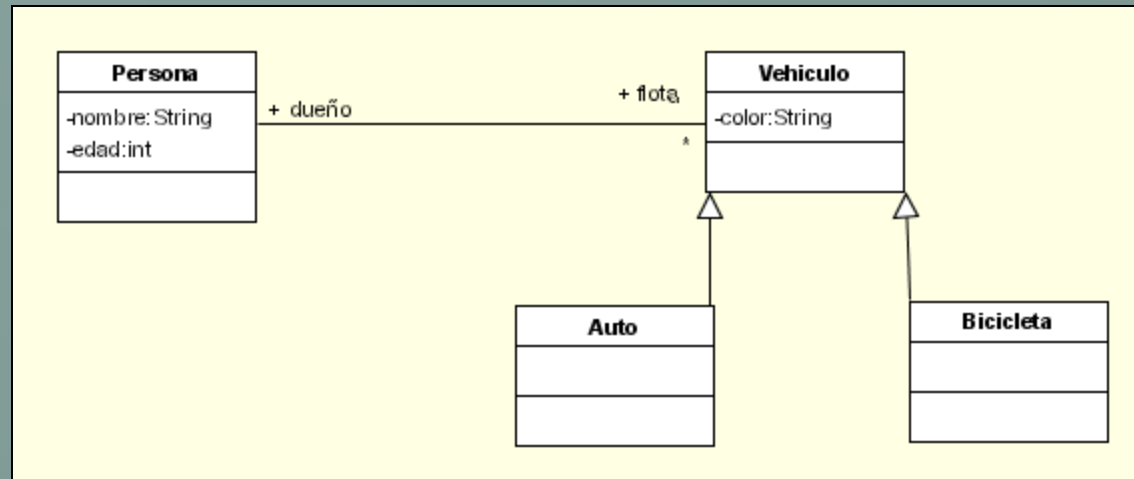
- Sus predicados “navegan” el diagrama de clases.
- “context” especifica el elemento del cual se está hablando.
- “self” indica el objeto actual.
- “result” representa el valor de retorno.
- OCL puede predicar sobre colecciones (conjuntos).
- Mediante “->” pueden utilizarse operaciones sobre colecciones (select, reject, exists, size, forAll, etc...).

# OCL: Object Constraint Language

- Un lenguaje formal textual para definir restricciones sobre objetos
- Usos
  - Modelo conceptual: Restricciones adicionales al modelo gráfico
  - Modelo de operaciones: Definición formal de condiciones (pre-, post- y disparadores)
  - Modelo de objetivos: Definición formal de objetivos (aunque habría que extenderlo con operadores temporales)
  - Diseño
    - Diagrama de Clases
    - Contratos
  - Navegación de modelos estructurales

# Repasando

“Todos los autos de una persona son negros”



context Persona

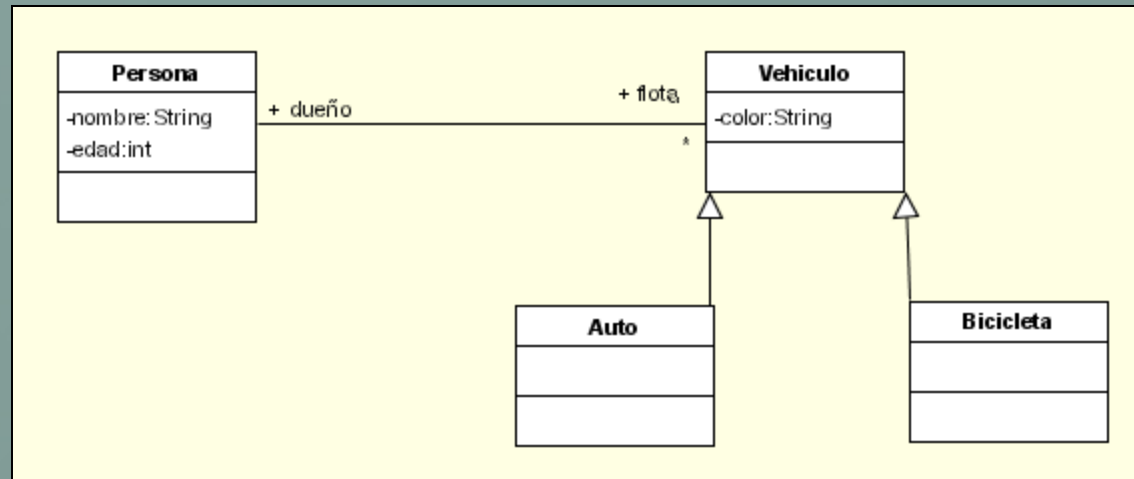
Conjunción

inv: self.flota->forAll(v | v.color="negro")



# Repasando

“Nadie tiene más de 3 vehículos negros”



context Persona

Conjunto  
de vehículos

inv: self.flota->select(v | v.color="negro")->size <= 3

# Visión Semántica

- Una descripción MC/OCL está dada en los mismos términos que una descripción de MC
  - Conjuntos de objetos con valores de atributos, relaciones entre conjuntos.
- La diferencia es la expresividad de OCL
  - Podemos **recortar** modelos inválidos con más precisión

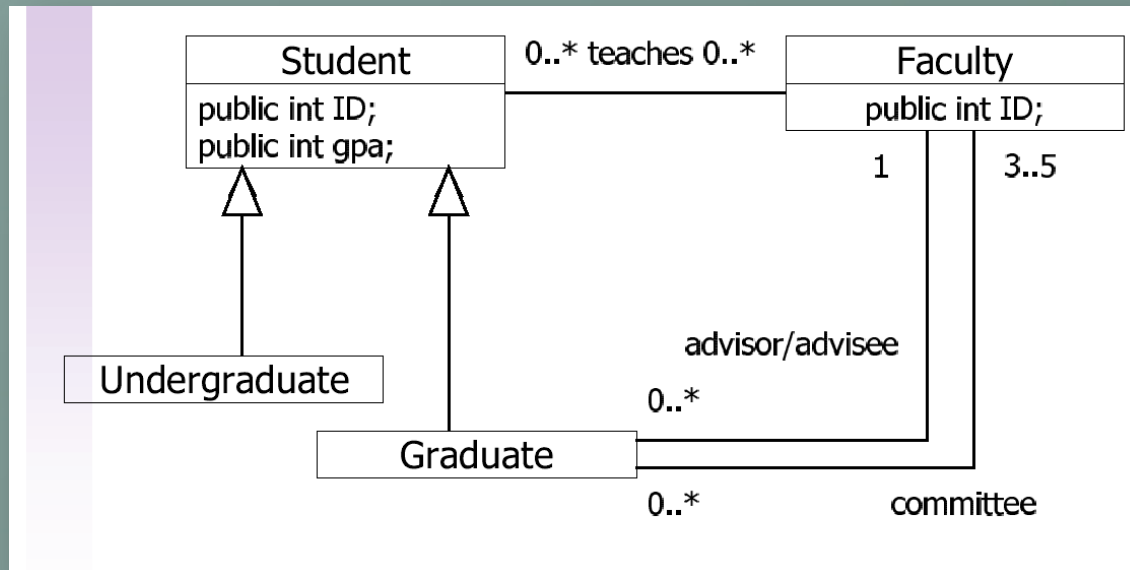


*Semántica de pruning o de recorte:  
a medida que agregamos especificación  
recortamos las interpretaciones  
válidas de la especificación*

*Modelos con semántica de pruning:  
Modelo de objetivos, de operaciones*

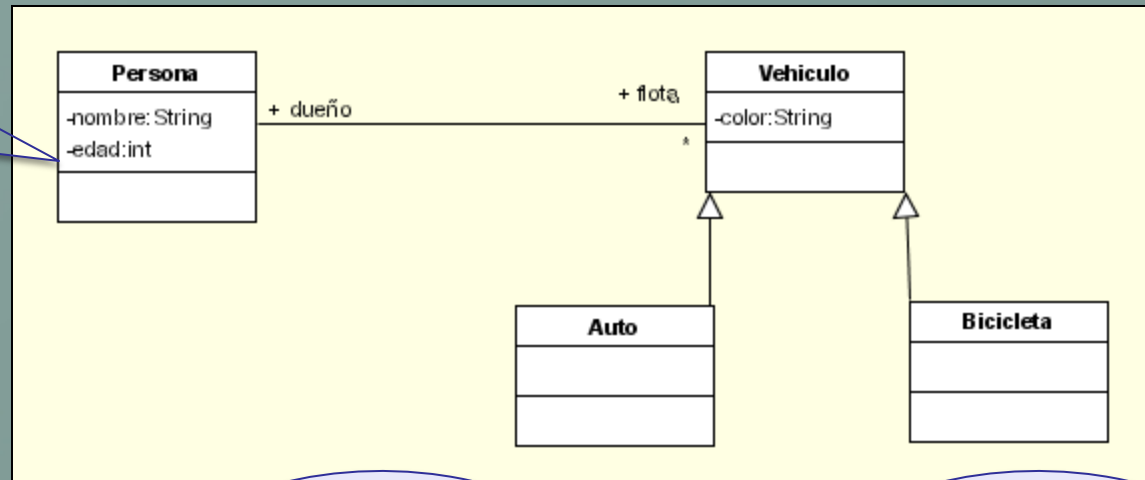
# Navegación por roles

- <Clase>.<rol> retorna una colección de objetos que están del otro lado de la asociación
  - Smith.teaches = {co842, sr771}
- Si multiplicidad = 1 entonces retorna un objeto
  - sr771.advisor/advisee = Smith



# Clases introducen Tipos en OCL

*Persona  
introduce un  
tipo nuevo a  
OCL*



*self es de tipo  
Persona*

Context Persona

*self.flota tiene  
tipo Colección  
de Vehículos*

*Devuelve el  
true si v es de  
tipo Auto*


inv: self.edad < 18 implies self.flota->forAll(v | not v.isKindOf(Auto))

# Colecciones

- OCL viene con tipos básicos **paramétricos** para modelar colecciones de objetos
  - **Collection(T)**: Colección genérica de elementos de tipo T
  - **Set(T)**: Colección de elementos de T, no ordenados, sin repetidos
  - **OrderedSet(T)**: Colección de elementos de T, ordenados, sin repetidos
  - **Bag(T)**: Colección de elementos, no ordenados, con repetidos
  - **Sequence(T)**: Colección de elementos, ordenados, con repetidos
- Supongamos **X e Y de tipo Collection(T)**, **t de tipo T** y **p(x): T-->Boolean**  
Estas son operaciones validas:
  - X->size()
  - X->intersection(Y), X->union(Y), ...
  - X->isEmpty(), X->notEmpty()
  - X->includes(t), X->excludes(t), X->count(t)
  - X->includesAll(Y), X->excludesAll(Y)
  - X->select(p(x)), X->forAll(p(x)), X->sum()
- **Set, orderedSet Bag y Sequence** son subtipos de Collection pero tienen especializaciones propias
  - Sequence y OrderedSet proveen first(), last()..
  - Bag provee count(t)...

# Reglas de Subtipificación

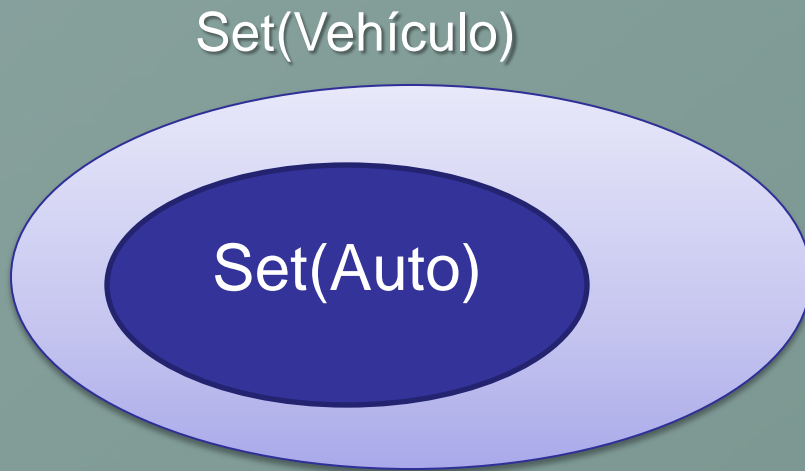
- Sean  $T1$ ,  $T2$  tipos correspondientes a clases  $T1$  y  $T2$ 
  - $T1 < T2$  sii  $T1$  es una subclase de  $T2$
  - Eg. Auto  $<$  Vehículo
- Para toda expresión de tipo  $T$ 
  - $\text{Set}(T) < \text{Collection}(T)$
  - $\text{Sequence}(T) < \text{Collection}(T)$
  - $\text{Bag}(T) < \text{Collection}(T)$
- Si  $T$  es un tipo entonces  $T < \text{OclAny}$
- La relación  $<$  es transitiva
- Si  $T1 < T2$  y  $C$  es Collection, Set, Bag o Sequence entonces  $C(T1) < C(T2)$



*¿Estará bien?*

# Subtipado de Colecciones

- Si  $T1 < T2$  y  $C$  es Collection, Set, Bag o Sequence entonces  $C(T1) < C(T2)$



Sean  $\text{Set}(\text{Auto}):X$  y  
 $\text{Set}(\text{Vehículo}):Y$

$X \rightarrow \text{union}(Y)$

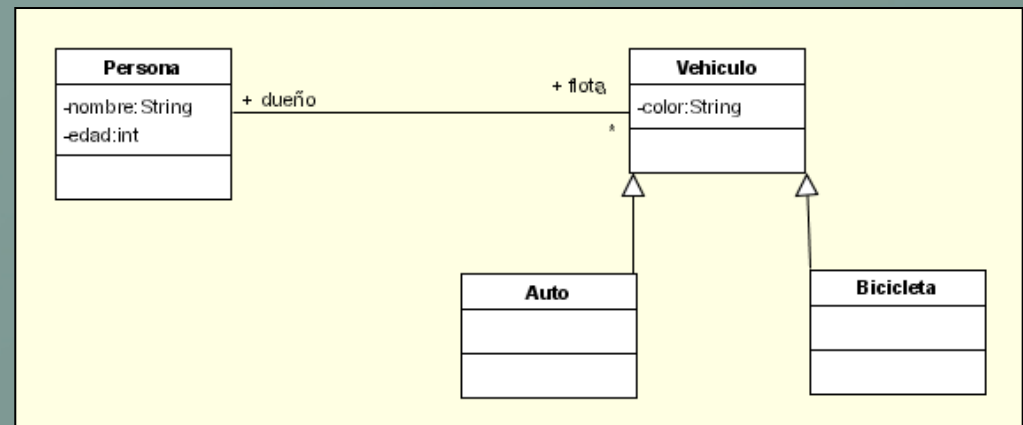
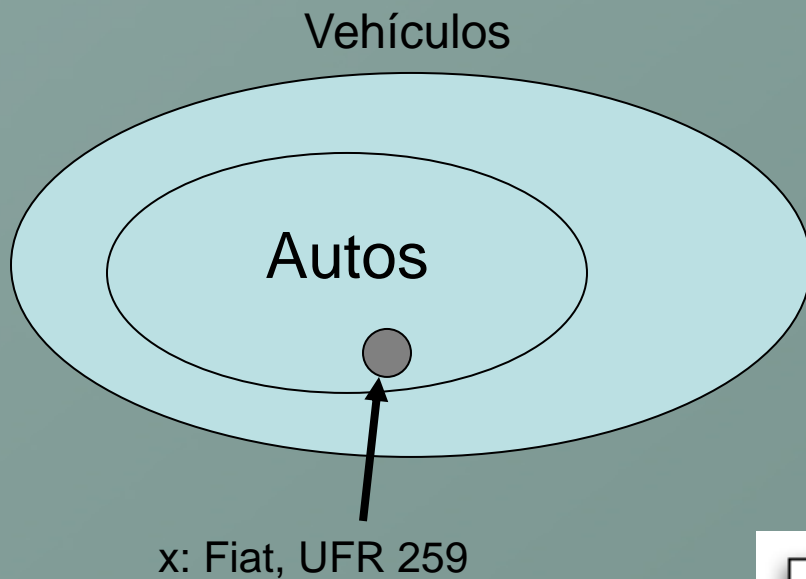
¿es una expresión permitida?

¿qué tipo tiene?

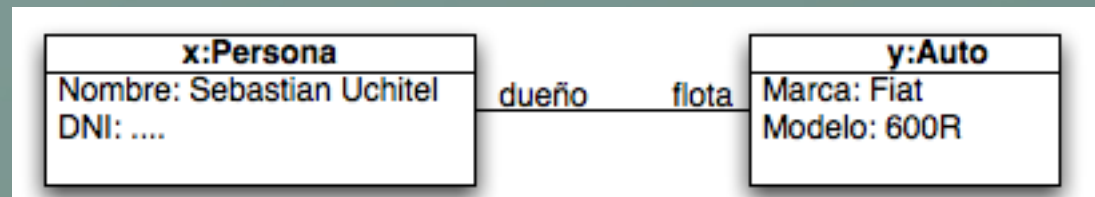
¿está bien pensar que  $X$  provee una  
operación unión?

# Tipos Aparentes y Reales

- El tipo aparente de una expresión es la que puede deducir del la signatura del diagrama de clases. Deducible estáticamente
- El tipo real se deduce del objeto mismo. Deducible dinámicamente



¿De qué tipo es x.flota?





# isTypeOf, isKindOf y asType

- `x.isKindOf(t)` retorna true si el tipo real de x es subtipo de t
- `x.isTypeOf(t)` retorna true si el tipo real de x es t
- `x.asType(t)` retorna una referencia (o variable anónima) denotando lo mismo que x pero con tipo aparente t

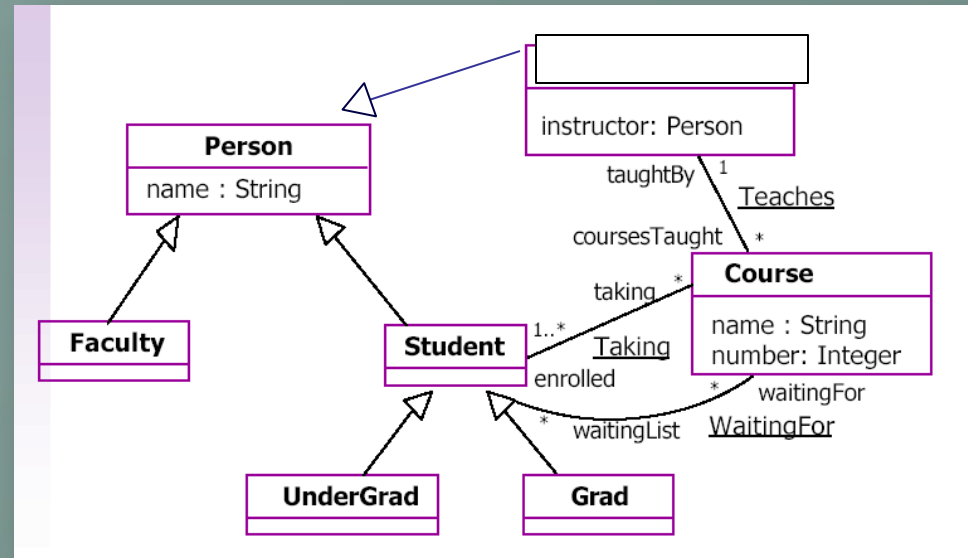
El proceso de  
cambiarle el tipo  
aparente a una variable  
(en OCL usando  
`asType`) se denomina  
casting

Context Instructor

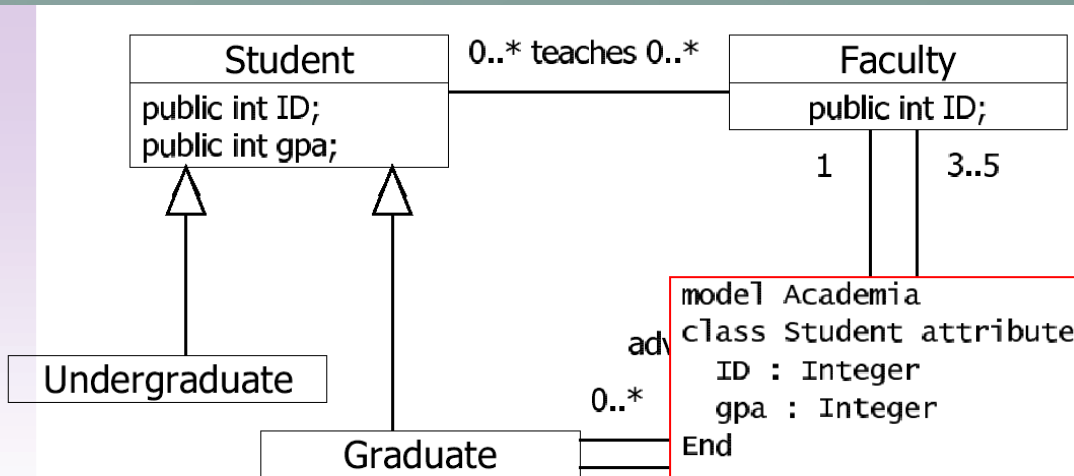
inv NoDictaSiCursa:

`self.isKindOf(Student) implies`

`self.coursesTaught->intersection (self.asType(Student). taking)->isEmpty()`



# Modelado con USE



```

model Academia
class Student attributes
  ID : Integer
  gpa : Integer
End

class Undergraduate < Student
class Graduate < Student

class Faculty attributes
  ID : Integer
End

association teaches between
  Student[*] Faculty[*]
End

association advisor between
  Graduate[*] Faculty[1]
end
  
```

```

association committee between
  Graduate[*] Faculty[3-5]
end

constraints

context Student
  inv gpaBound: self.gpa >= 0 &
    self.gpa <= 4
  inv uniqueID:
    Student.allInstances->forAll(
      s1, s2 |
        s1!=s2 implies s1.ID != s2.ID)
end

context Graduate
  inv advonCommittee:
    self.committee->includesAll(
      self.advisee)
end
  
```

# OCL: Resumen

- Especificación formal de invariantes de los objetos
- Expresiones OCL **utilizan vocabulario** del modelo conceptual.
  - El MC introduce una signatura para las aserciones OCL
- Predicados “navegan” el diagrama de clases.
- **context** especifica el elemento del cual se está hablando.
- **self** indica el objeto actual.
- Soporte para predicar sobre colecciones (“->”)
  - **select, exists, size, forAll**, etc...

# Relación con Otros Modelos

- OCL y el Modelo Conceptual (Clases y Objetos)
  - Están intrínsecamente relacionados
  - OCL permite mayor expresividad respecto a los estado válidos del mundo
- OCL y Objetivos
  - OCL puede utilizarse para formalizar los objetivos
  - Context t:Trains  
inv PuertaSoloSiVelocidad0yEnPlataforma  
t.velocidad=0 AND  
Platform->allInstances->select(p | p=t.position) -> notEmpty)
- OCL y el modelo de Operaciones
  - OCL puede ser el lenguaje para formalizar operaciones provistas por la maquina y otros agentes
- OCL y el modelo de Contexto
  - Descripción más refinada del estado de los agentes

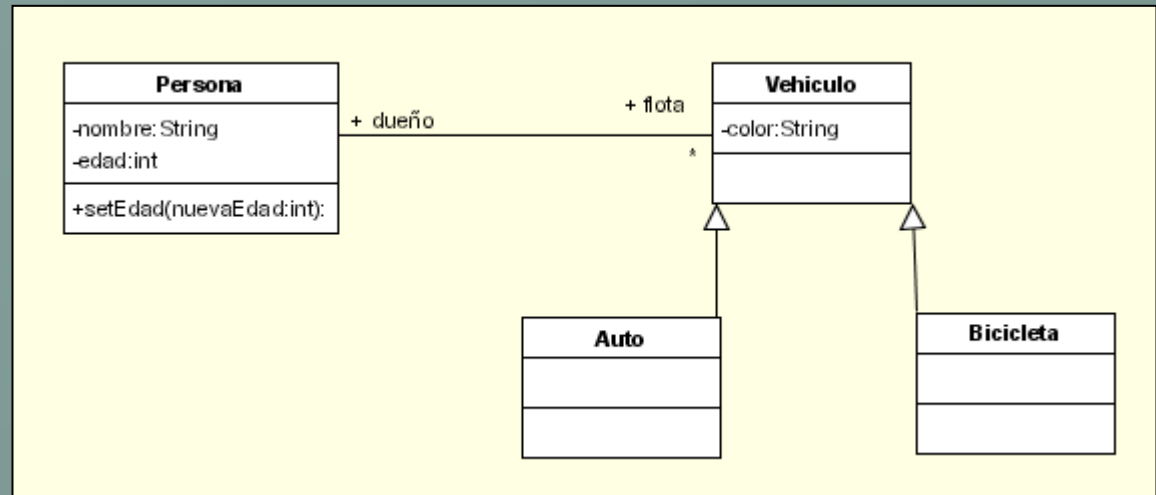
# OCL por fuera del Modelo Conceptual

- Por completitud, veamos usos de OCL por fuera del MC.
- No vamos a utilizar estas formas en la materia....

# OCL para Modelo de Operaciones

Puede utilizarse para definir precondiciones y postcondiciones de operaciones (o casos de uso!)

Modelo de Operaciones



setEdad(p:Persona, nuevaEdad:int)

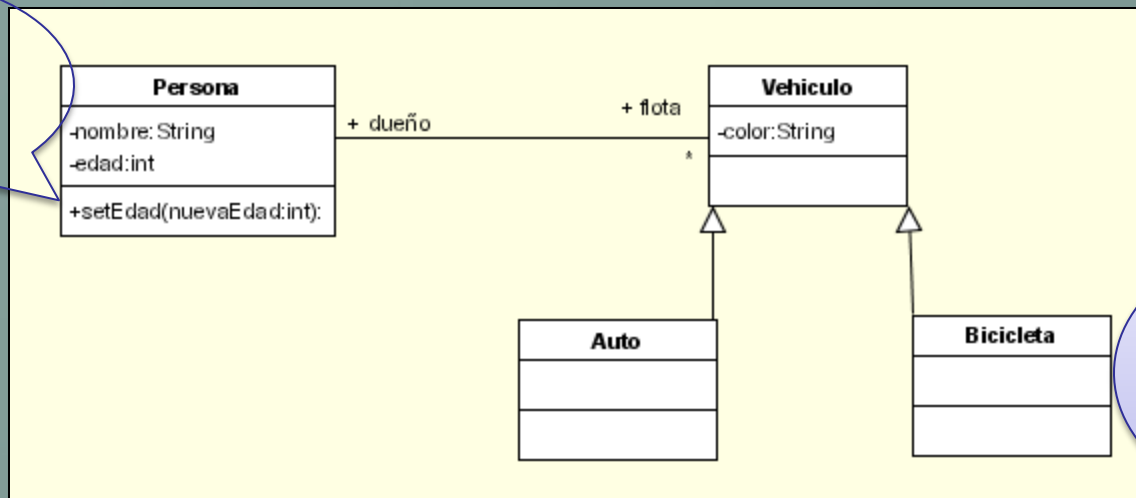
pre: p.nuevaEdad >= 0

post: p.edad = nuevaEdad

# OCL para Diseño

Puede utilizarse para definir precondiciones y postcondiciones de operaciones (o casos de uso!)

*Orientado a  
Diseño*




*Orientado a  
Diseño*

```
context Persona::setEdad(nuevaEdad:int)
pre: nuevaEdad >= 0
post: self.edad = nuevaEdad
```

# Operador @pre

## *Student::dropCourse(c: Course)*

```
context Student::dropCourse(c: Course)
pre  NowTaking:  taking->includes(c)
post NotTaking:  taking = taking@pre->excluding(c)
```



*Yields value of 'taking' in the pre-state*



# Ejemplo de Pre/Post

## *Sorting sequences of integers*

---

```
context A::sort(s : Sequence(Integer)) : Sequence(Integer)

  post SameSize:
    result->size = s->size

  post SameElements:
    result->forall(i | result->count(i) = s->count(i))

  post IsSorted:
    Sequence{1..(result->size-1)}->
      forall(i | result.at(i) <= result.at(i+1))
```

# Operadores Para Post-Condiciones

```
model Graph
```

```
class Node
operations
  newTarget()
end
```

```
association Edge between
  Node[*] role source
  Node[*] role target
end
```

```
constraints
```

```
context Node::newTarget()
-- the operation must link exactly one target node
post oneNewTarget:
  (target - target@pre)->size() = 1

-- the target node must not exist before
post targetNodeIsNew:
  (target - target@pre)->forall(n | n.IsNew())
```

- ***o.IsNew***: True sii o no existía en el estado anterior a la aplicación de la operación