diseño de sistemas

utn frlp

U3 – OCL (Object Constrain Language)

Temario

- ¿Qué es OCL?
- Motivación de su implementación.
- Contexto de la expresión, aplicación.
- Etiquetas de expresión:
  - *Invariantes*
  - *Pre / post condiciones*
  - *Inicialización*
  - *Derivación*
  - *De consulta*
- Navegación.
- Manejo de colecciones.

utn frlp ds 

OCL - Presentación

- Es un lenguaje formal.
- Describe expresiones sobre modelos UML.
- Forma parte del estándar UML, fue desarrollado en IBM.
- La evaluación de expresiones OCL *no afecta* el estado del sistema en ejecución.
- Es un lenguaje *declarativo* (qué y no cómo) y *tipado*.
- Complementa los modelos para refinarlos, representando la realidad en forma más precisa.
- No es un lenguaje de programación sino un lenguaje de especificación.
- Es un lenguaje de expresión puro. No modifica el estado ni la estructura del modelo.

utn frlp ds 

Motivación

- Lenguaje natural conduce a ambigüedades.
- Los métodos formales no son usados masivamente.
- UML no tiene expresividad suficiente para precisar ciertas restricciones.
- Pasar de expresiones en notas a expresiones más formales y precisas.
- Especificar condiciones para lograr modelos bien formados.
- Posibilita expresar restricciones semánticas de un sistema, que no pueden expresarse de otra manera.

utn frlp ds 

Aplicación

- La entidad de aplicación se llama *clasificador* y puede ser una clase, una interfaz, atributos de una clase, un componente.
- Puede usarse como lenguaje de consulta.
- Para expresar condiciones invariantes en clases y tipos.
- Para definir pre y post condiciones en operaciones de clases e implementación de métodos en interfaces.
- Especificar reglas del negocio.
- Especificar condiciones para un modelo bien formado.
- Para escribir guardas (Diag. Estados).
- Para especificar condiciones (Diag. Secuencia).
- Permite definir el cuerpo de operaciones de consultas.
- Para expresar reglas de derivación.

utn frlp ds 

Síntesis

Las expresiones no ambiguas → modelos más coherentes y detallados.

(Diagramas UML + restricciones OCL) = Modelos precisos - consistentes

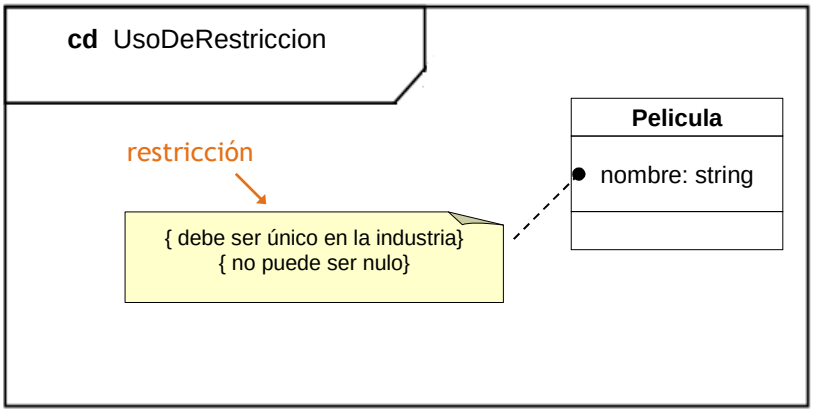
Toda expresión OCL se basa en tipos primitivos y tipos definidos por el *clasificador* UML:

CLASE – INTERFAZ – ESTADO - COMPONENTE

utn frlp ds 

Motivación

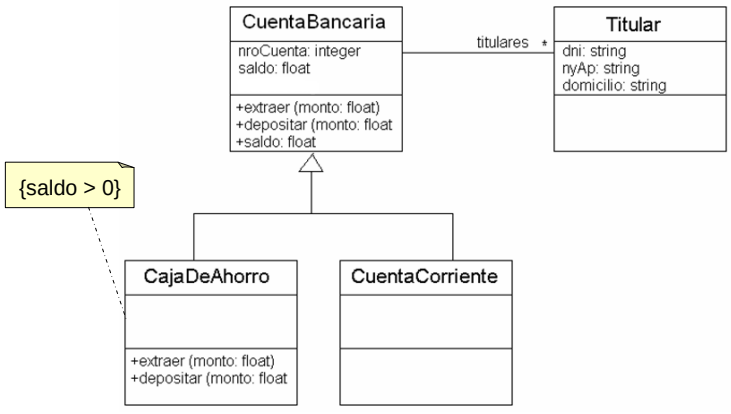
Necesidad de restricciones más formales…



utn frlp ds

Motivación

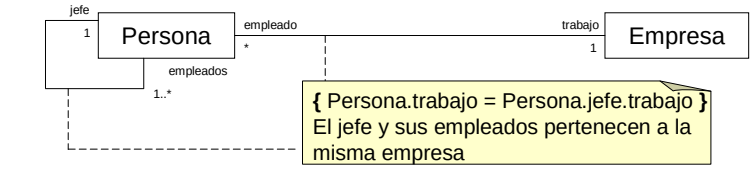
Necesidad de restricciones más formales…

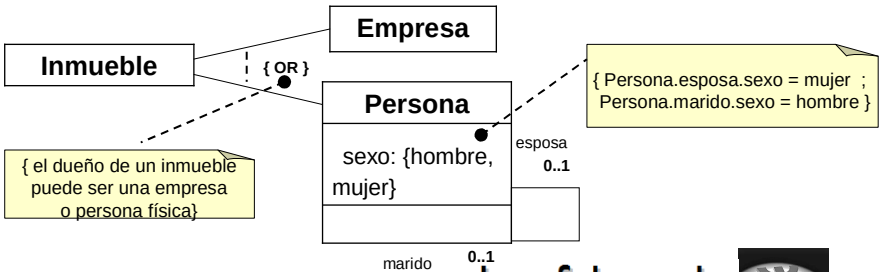


utn frlp ds

Motivación

Necesidad de restricciones más formales…

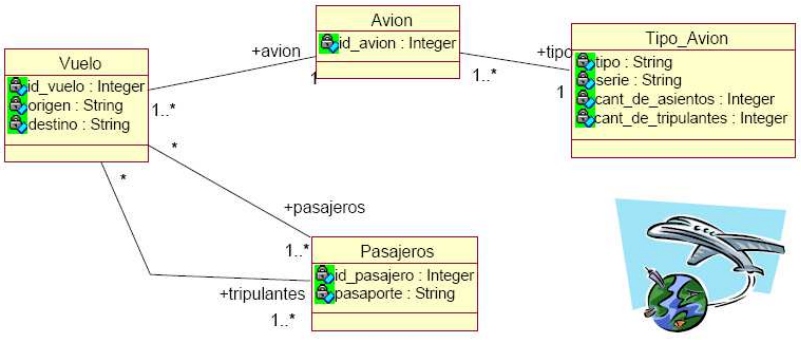




utn frlp ds

Motivación

Multiplicidad dinámica

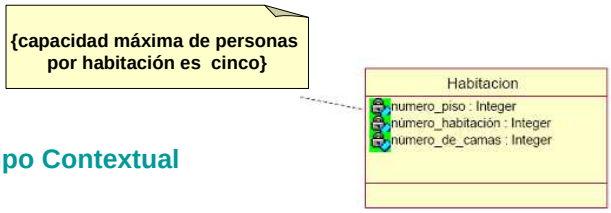


Controlar la venta de pasajes respecto de la capacidad del avión.

utn frlp ds

Contexto de una expresión

El vínculo entre una entidad de un diagrama UML y la expresión OCL define el ámbito de aplicación.



Tipo Contextual

```
context Habitacion
inv:self.numero_de_camas <= 5
```

Instancia Contextual

Etiqueta

utn frlp ds

Contexto de una expresión

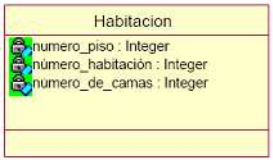
También se puede ampliar el contexto describiendo el modelo, paquete de pertenencia del clasificador y un nombre a la expresión.


```
modelo ModeloAdmHotel
  paquete GestionReservas
    context Habitacion
      inv CupoMaximo: self.numero_de_camas <= 5
  finPaquete
finModelo
```

utn frlp ds

Invariante

Una expresión con etiqueta invariante INV es una condición que deberá ser verdadera para todas las instancias del tipo específico, clasificador (Ej: clase Habitación de un Hotel que tiene 3 pisos)



context Habitacion
inv:self.numero_piso > 0 and self.numero_piso <= 3

context Habitacion
inv: numero_piso > 0 and numero_piso <= 3

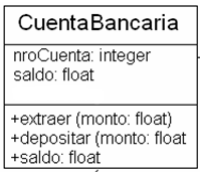
context h:Habitacion
inv: h.numero_piso > 0 and h.numero_piso <= 3

Tres formas distintas de escribir la expresión

utn frlp ds

Pre y Post Condiciones

Ejemplo:



context CuentaBancaria::extraer(monto float)
pre:monto > 0
pre:self.saldo >= monto
post: self.saldo = self.saldo@pre -monto

En el Diseño por **Contratos** se especifican las **Responsabilidades** de las clases. Un objeto brinda servicios (obligaciones) en base a condiciones que debe alcanzar (derechos).

Derechos ⟶ precondiciones
Obligaciones ⟶ postcondiciones

utn frlp ds

Pre y Post Condiciones

Una expresión con etiqueta PRE o POST representa una condición que debe ser verdadera antes (pre) y después (post) de ejecutar una operación, → su contexto de aplicación es una operación, deberá incluir la clase y la operación (signatura). Post especifica valor de retorno por medio de la palabra reservada RESULT

Forma genérica de la expresión:

context Clase::operación (param1 : tipo1, ….) :tipoRetorno
pre: (condición de param1)
post: (condición)
post: result = (objeto retorno)

Nota: solo en una postcondición se puede hacer referencia al valor inicial de algún atributo o parámetro agregando @pre al final del elemento en cuestión.

utn frlp ds

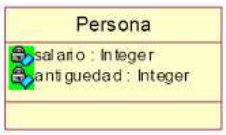
Valores Iniciales y Derivados

Una expresión con etiqueta INIT establece el valor inicial de un atributo o extremo de una asociación.

Ejemplo: El salario inicial de todo empleado es 5000 $.

¿Cuál es el tipo contextual? ¿Cómo definimos la instancia contextual?

context Persona::salario Integer
init: 5000



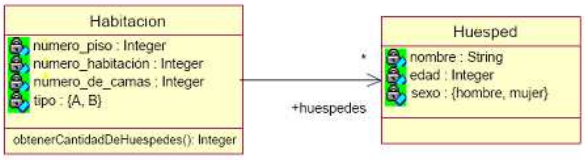
Ejemplo: El salario se incrementa en 120 $ por cada año de antigüedad

context Persona::salario Integer
derive: self.salario + (120 * self.antiguedad)

utn frlp ds

Consulta

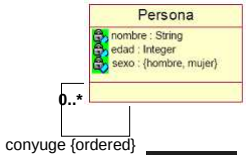
Una expresión etiquetada como BODY nos indica el resultado de una operación de consulta. Sirve como implementación preliminar de un método.



context Habitacion::obtenerCantidadDeHuespedes() :Integer
body: self.huespedes -> size()

isQuery

context Persona::nombreParejaActual() : String
body: if self.conyuge -> notEmpty() then self.conyuge ->
last().nombre
else null
endif



utn frlp ds

Más expresiones… Let y Def

Usamos LET definir una variable local con validez es el context, mientras que DEF define un atributo derivado y define la regla de derivación.

context Persona
let: hoy : Date = now()
def: edad :Integer = (hoy – self.fechaNacimiento).mod(365)

Observación: como resultado de una navegación un objeto puede ser tratado como una colección de dimensión 1 . Ej: Persona tiene 0..1 conyuge

context Persona
inv: self.conyuge -> isEmpty() and self.sexo = Sexo::hombre implies “No tiene esposa a cargo”

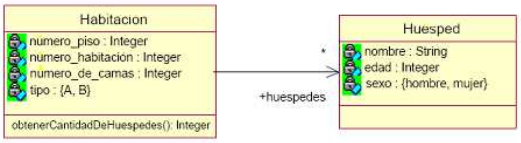
Otra forma: inv: self.conyuge.isEmpty()
……..

utn frlp ds

Navegaciones

Permite referirse a objetos relacionados a través de asociaciones

Navegaciones simples y combinadas.



Simple: se navega a través de una asociación. Ej: Cantidad de huéspedes alojados en una habitación no debe ser mayor que su capacidad.

```
context Habitacion
inv: self.huespedes->size() <= self.numero_de_camas
```

Ojo! Con la conformidad de tipos

utn frlp ds

Colecciones

La colección es un tipo abstracto.

Los tipos concretos son:

SET	Conjunto
ORDEREDSET	Conjunto ordenado
BAG	Bolsa (admite repetición)
SEQUENCE	Subconjunto de un conjunto ordenado

Operaciones de colecciones:

Size, select, reject (not), collect, forAll, exists, empty, notEmpty, count(), sum(), Include(), includeAll, iterate, union

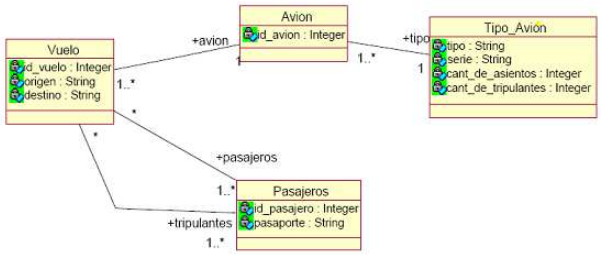
Cada sub-clase tiene operaciones especificas

utn frlp ds

Navegaciones combinadas

Navegaciones combinadas (encadenadas) admiten varios niveles acceso, a través de varias asociaciones.

Ejemplo: Controlar la cantidad de pasajeros de acuerdo a la capacidad del avión. (Multiplicidad dinámica)



context Vuelo
inv: self.pasajeros->size() <= self.avion.tipo.cant_de_asientos

utn frlp ds

Colecciones y Navegaciones combinadas

Los tipos de las colecciones deben ser tenidos en cuenta ya que son importantes, pues se relacionan con el resultado de navegaciones.

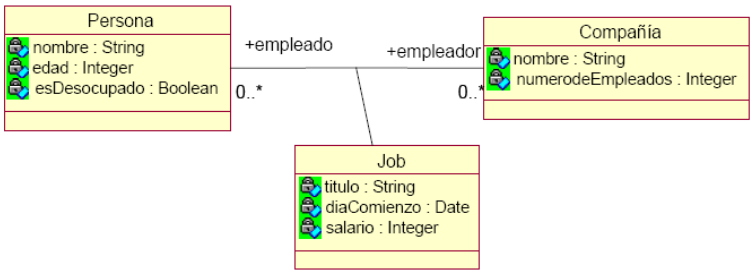
- Navegación simple y la multiplicidad de la asociación es mayor a 1 (SET)
- Navegaciones combinadas resultan en BAG si al menos en una de las asociaciones la multiplicidad es mayor a 1.
- Navegación simple en asociación adornada, etiquetada {ordered} resulta en ORDEREDSET
- Navegaciones combinadas en asociaciones adornadas, etiquetadas {ordered} resultan en SEQUENCE

Las navegaciones definen acoplamiento de objetos

utn frlp ds

Navegaciones y clases de asociación

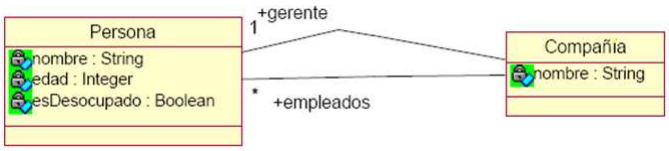
Se puede navegar desde una clase de asociación a los objetos que participan de la asociación



context Job
inv: self.empleado.edad >= 21

utn frlp ds

Más navegaciones…



Ejemplo: Toda compañía tiene 1 gerente que esta ocupado.
Toda compañía debe tener empleados (al menos 1)

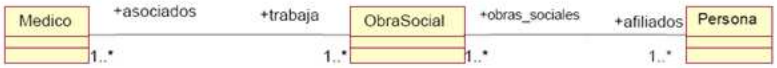
context Compañia
inv: self.gerente->size()=1 implies self.gerente.esDesocupado = false

inv:self.empleados->notEmpty /*otra forma inv: self.empleados->size() > 1 */

utn frlp ds

Más navegaciones…

¿Cuándo y por qué las navegaciones combinadas resultan en un BAG?



Ejemplo: un medico quiere dimensionar los pacientes potenciales de las obras sociales 'IOMA' y 'PAMI' en las cuales recientemente se ha asociado.

```
context Medico::pacientesPotenciales() :Integer
body: self.trabaja->select (o / o.name ='‘IOMA' or o.name=
'PAMI´).afiliados ->count()
```

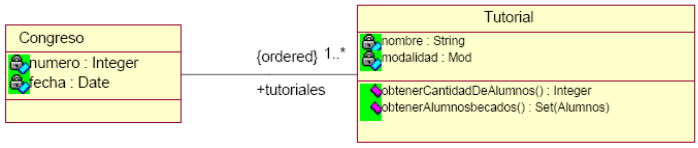
Puede haber duplicaciones

```
context Medico::pacientesPotenciales() :Integer
body: self.trabaja->select (o / o.name ='‘IOMA' or o.name=
'PAMI´).afiliados ->asSet()->count()
```

Analizamos los problemas de la navegación utn frlp ds

Más navegaciones…

En asociaciones etiquetadas ordered resultan ORDEREDSET Y SEQUENCE
Ejemplo: podría consultar matrícula del primer tutorial, el cuarto y el ultimo y sumarlas.

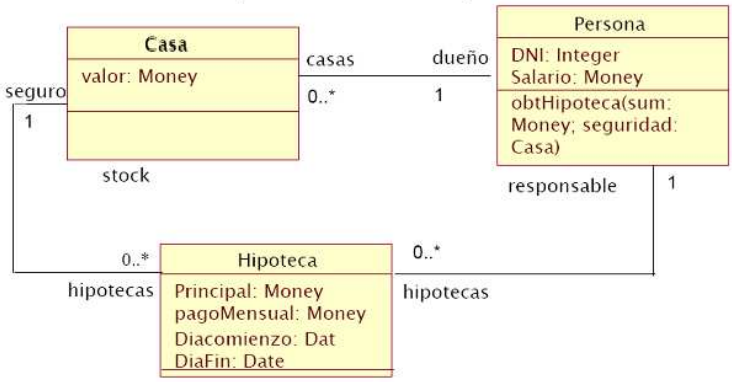



```
context Congreso::sumarMatricula() :Integer
body: self.tutoriales->first().obtenerCantidadDeAlumnos() +
self.tutoriales->at(4).obtenerCantidadDeAlumnos() +
self.tutoriales->last().obtenerCantidadDeAlumnos()
```

```
context Congreso::buscarTutoriales(unaModalidad :Mod) :Collection
…. self.tutoriales->select( t / t.modalidad = unaModalidad)
```
(Resulta una secuencia)

Analizamos los problemas de la navegación utn frlp ds

Multiplicidad Opcional

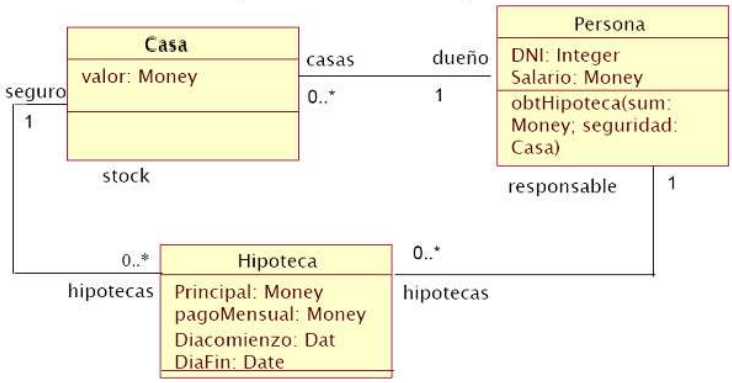


Especificar en OCL la siguiente restricción: Una persona es responsable de hipotecas de sus propias casas. Sino tiene casa tampoco hipoteca.
Una posible solución….*Construimos grupalmente otra alternativa*

```
context Persona
inv: self.hipotecas->exists() implies self.casas->notEmpty()
```

utn frlp ds

Multiplicidad Opcional



Context Persona
Inv: self.casas includesAll(self.hipotecas.seguro)

utn frlp ds

Cuadro Resumen

Context Clasificador
Inv: (condición)

context Clase::operación (param1 : tipo1, ….) :tipoRetorno
pre: (condición de param1)
post: (condición) o post: result = (objeto retorno)

context Clasificador::atributo :Tipo
init: valor

derive: (expresión – fórmula)

context Clase::operación (param1 : tipo1, ….) :tipoRetorno
body: (consulta que devuelve)

let: vblelocal: tipo = valor
def: atributo derivado (regla de derivación)

coleccion -> **select**(expresion, condición booleana) - obtiene una subcoleccion

coleccion-> **isEmpty**() (booleano)

Colección->**Collect**(tipoObjeto) – obtiene una colección de otra con distinta composicion

unBag->**asSet**() convertirlo a set (sin repetidos)

colección->**forAll**(h: Clase / condición con h) (booleano)

Condición1 **implies** (afirmación o condición true)

(al menos uno de una coleccion cumple una condicion) colección ->**exists**(condicion)

Colección->**count**()
Colección->**sum**()
Coleccion1-> **includes**(objeto)
Coleccion1-> **includesAll**(coleccion2)
At(index), First(), Last()

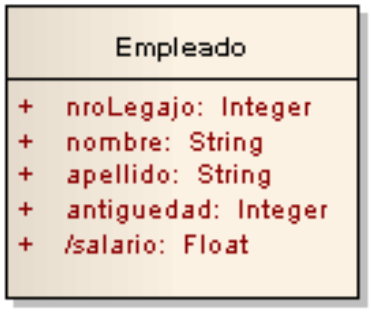
utn frlp ds

Casos Prácticos

Aplicamos restricciones OCL que consideremos a los siguientes modelo (clase UML) explicitando:

utn frlp ds

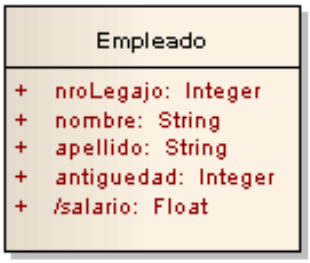
a) La variable de instancia antigüedad nunca puede tomar valor negativo.



{ context Empleado
inv: self.antiguedad > =0 }

utn frlp ds

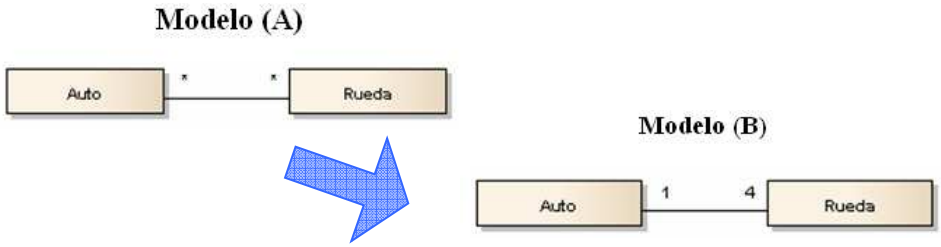
b) El valor de la variable de instancia salario es el importe resultante de un monto inicial de $ 4500 más $ 300 por cada año cumplido de antigüedad.



{ context Empleado :: salario: Float
derive: 4500 + (300 * self.antiguedad)}

¿Por qué no usamos INIT ? Donde se guardaría el valor inicial?

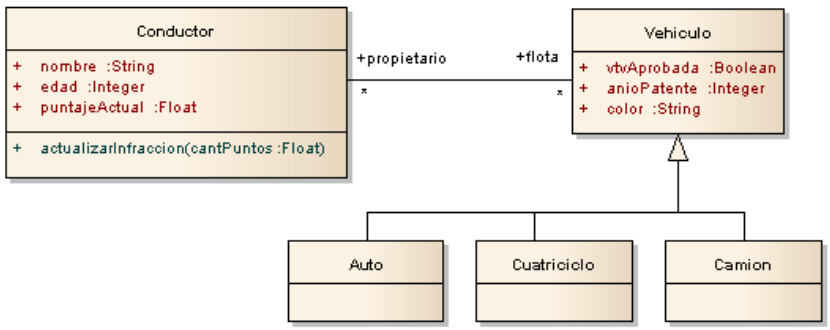
utn frlp ds



{ context Auto
inv: self.rueda->size() = 4 }
-- todo Auto se relaciona con exactamente 4 instancias de Rueda.

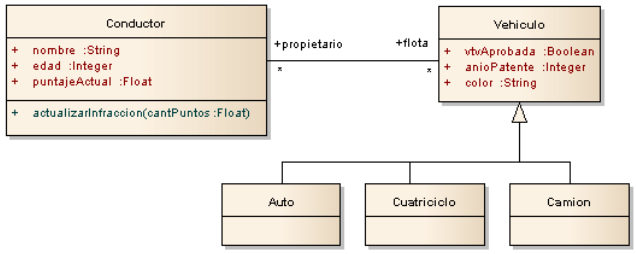
{ context Rueda
inv: self.auto is ->size() = 1 }
-- toda Rueda se relaciona con exactamente 1 instancia de Auto.

c) Todo Conductor posee mínimamente un vehiculo, pero no más de tres.



{ context Conductor
inv: self.flota -> size() >= 1 AND self.flota -> size() =< 3 }

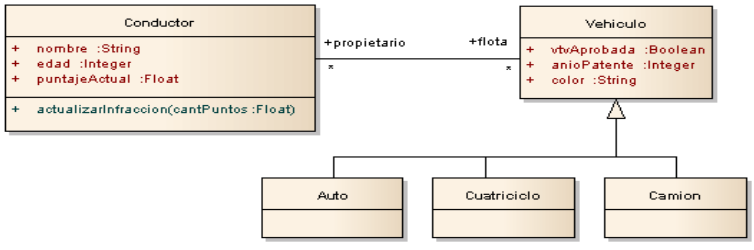
d) Todo Conductor debe tener al menos 18 años de edad y no debe tener puntaje con valor negativo. Los que son dueño de un Camión debe tener mínimamente 21 años de edad.



{ context Conductor
inv: self.edad >= 18 AND self.puntajeActual > 0 }

{ context Camion
inv: self.propietario.edad >= 21 }

e) Todos los Vehiculos de un Conductor deben tener: año de patentamiento mayor a 2010 y aprobada la Verificación Técnica Vehicular (VTV). Resolver usando “forAll”.



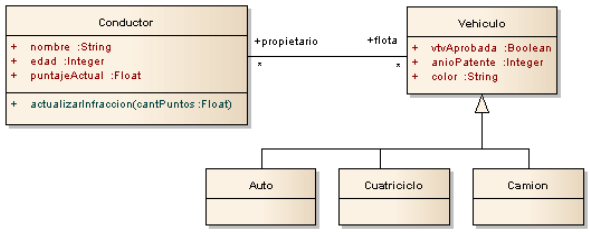
{ context Conductor
inv: self.flota -> forAll (v: Vehiculo | v.anioPatente > 2010 AND v.vtvAprobada = 'true') }

{ context Vehiculo
inv: self.propietario.size()>0 and self.anioPatente > 2010 AND self.vtvAprobada = 'true') }

f) Explicitar al método actualizarInfraccion() de la clase Conductor:

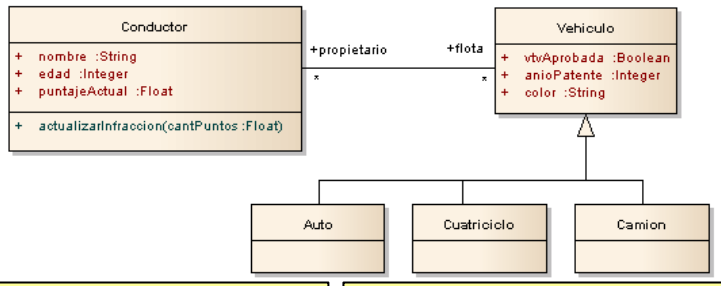
pre-condición --> *El puntaje (atributo puntajeActual) del Conductor nunca será negativo ni se actualizará con puntos (parámetro cantPuntos) de valores negativos.*

post-condición --> *El puntaje (atributo puntajeActual) del Conductor se verá decrementado en cantPuntos*



{ context Conductor :: actualizarInfraccion(cantPuntos: Float)
pre: cantPuntos > 0
pre: self.puntajeActual >= cantPuntos
post: self.puntajeActual = self.puntajeActual@pre - cantPuntos }

g) Un Cuatriciclo debe tener obligatoriamente como propietario un único conductor, mientras que los Autos y Camiones deben tener obligatoriamente como propietario al menos dos conductores (un Titular y Cotitular).



{ context Cuatriciclo
inv: self.propietario -> size() = 1 }

{ context Auto
inv: self.propietario -> size() > =2 }

{ context Camion
inv: self.propietario -> size() >= 2 }

Bibliografía

- Referencia oficial en OMG:

  www.omg.org

  http://www.omg.org/spec/OCL/

  The object Constraint Language – Getting your models ready for MDA - Second Edition – Jos Warmer, Anneke Kleppe (Addison Wesley)

utn frlp ds