Memoria Práctica 1 - Modelado estructural de un sistema de aviación

Realizado por Manuel Fuentes Vida, Clemente Cano Mengíbar, Marcos Díaz Sánchez, María Victoria Huesca Peláez, Alejandro López Ortega, David Luque Molina, José Antonio Casado Molina.

(Grupo 1-2)

Contenido

Diagramas de clases

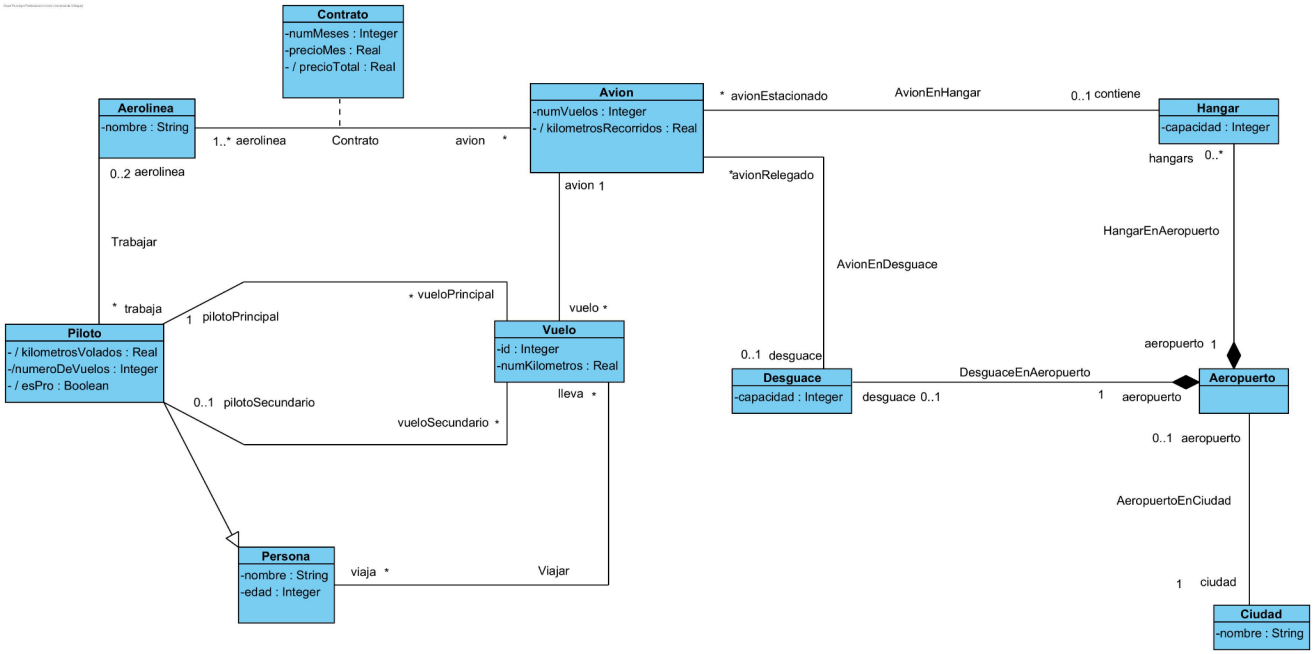


Diagrama de clases en Visual Paradigm

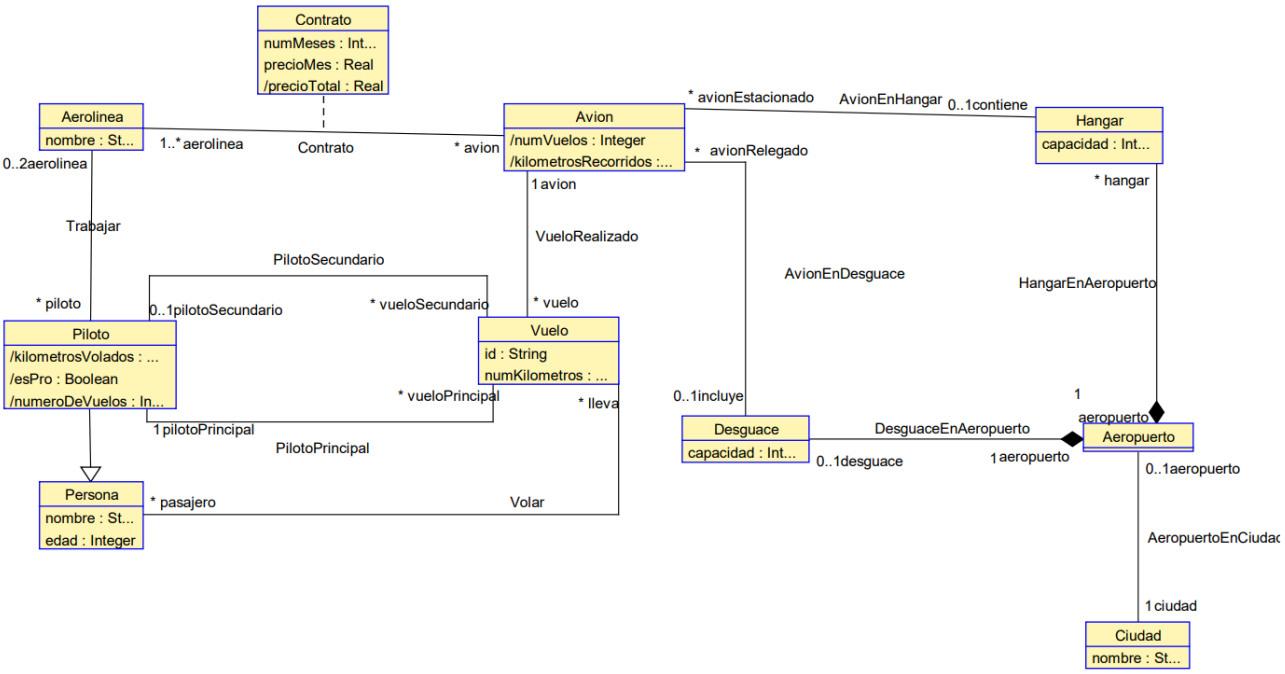


Diagrama de clases en USE

Diagrama de objetos

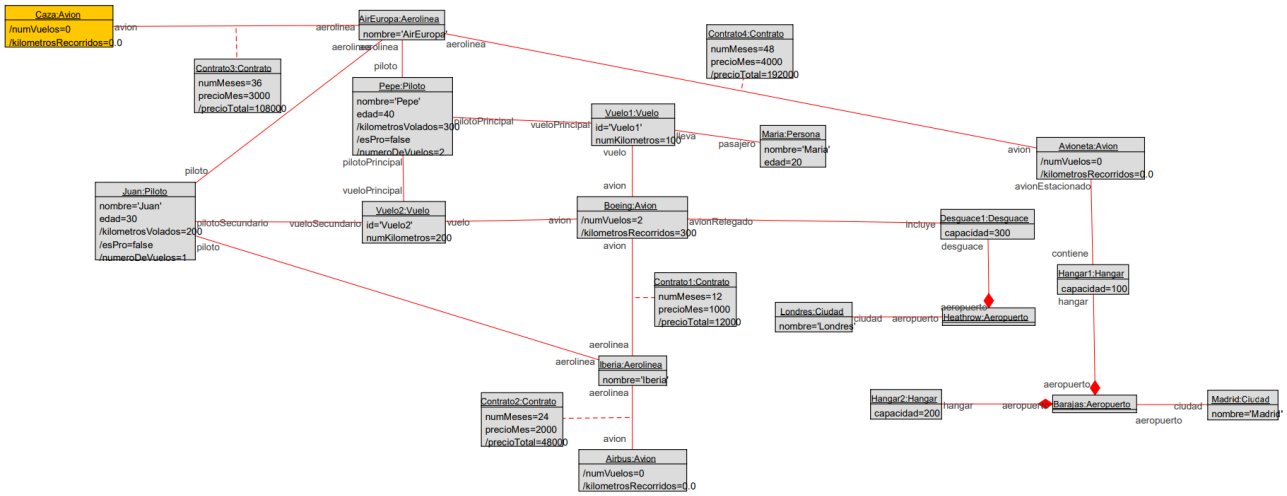


Diagrama de objetos número 1

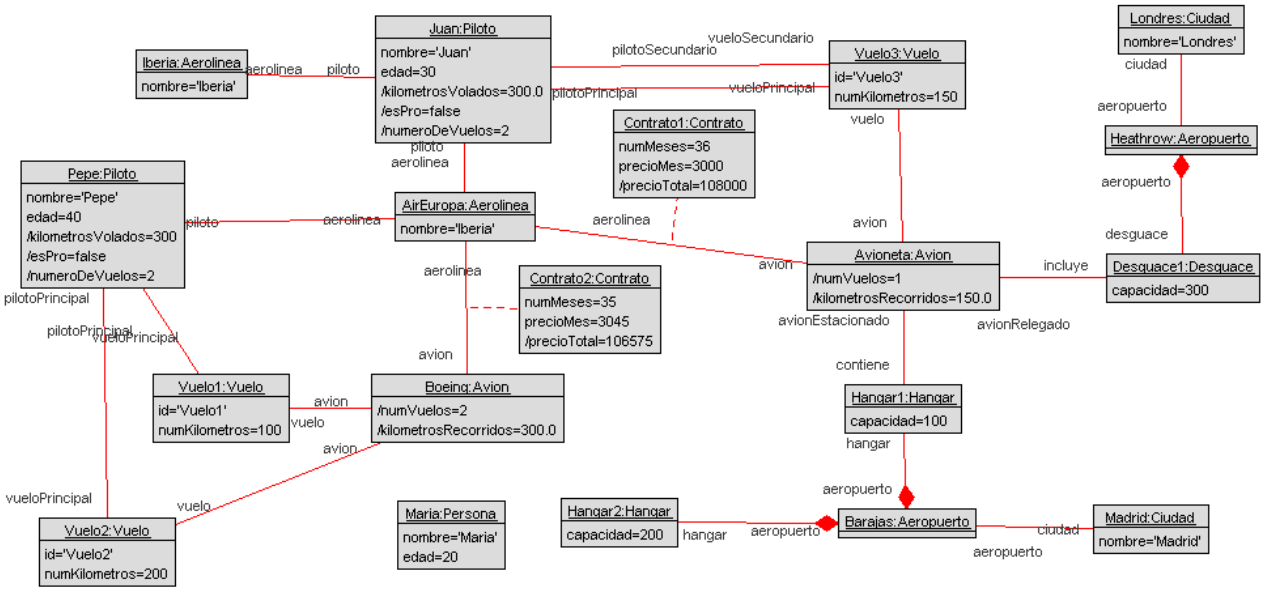


Diagrama de objetos número 2

Restricciones definidas

Las restricciones definidas son:

Aerolínea

-NombreDistinto:

```
-- Las aerolíenas deben tener nombres distintos
inv nombreDistinto :
  Aerolinea.allInstances()->forAll(a1, a2 | a1 <> a2 implies a1.nombre <> a2.nombre)
```

En esta restricción hemos definido que no haya 2 aerolíneas con el mismo nombre

Avión

-HangarODesguace:

```
-- Un avion no puede estar en un hangar y en un desguace a la vez
inv SiEstaEnHangarNoEstaEnDesguace :
  not (self.contiene->notEmpty() and self.incluye->notEmpty())
```

En esta restricción definimos que si un avión se encuentra en un desguace, debemos tener en cuenta que no se encuentre en un hangar y viceversa.

Desguace

-MasDe1000Vuelos:

```
-- Los aviones en desguace deben tener más de 1000 vuelos
inv avionEnDesguace1000Viajes :
  self.avionRelegado->forAll(a | a.numVuelos >= 1000)
```

Nos aseguramos de que ningun avion que se encuentre en un desguace tenga menos de 1000 vuelos

-**CapacidadDesguace:**

```
-- Los desguaces no pueden tener más aviones de su capacidad
inv avionesEnDesguaceMenorACapacidad :
  self.avionRelegado->size() <= self.capacidad
```

Comprobamos que el número de aviones almacenados en el desguace no supere su capacidad máxima

Hangar

-CapacidadHangar:

```
-- Los hangares no pueden tener más aviones de su capacidad
inv avionesEstacionadosMenorACapacidad :
  self.avionEstacionado->size() <= self.capacidad
```

Comprobamos que el número de aviones almacenados en el desguace no supere su capacidad máxima.

Vuelo

-PilotosDistintos

```
-- Los pilotos principal y secundario deben ser personas distintas
inv pilotosDistintos :
  self.pilotoPrincipal <> self.pilotoSecundario
inv identificadorUnico :
  Vuelo.allInstances()->forAll(v1, v2 | v1 <> v2 implies v1.id <> v2.id)
```

Hemos de tener en cuenta que en un mismo vuelo, el piloto principal y el secundario no sean la misma persona, asegurando que todos los pilotos tengan ids distintos

Escenarios

Primer escenario

En este escenario se comprueba que se cumplen todos los invariantes que se han establecido excepto el que comprueba que un desguace sólo contenga aviones con 1000 vuelos o más para el que hemos realizado una versión del invariante con sólo 2 vuelos.

Class invariants

Invariant	Satisfied
Aerolinea::nombreDistinto	true
Avion::SiEstaEnHangarNoEstaEnDesguace	true
Desguace::avionEnDesguace1000Viajes	false
Desguace::avionEnDesguace2Viajes	true
Desguace::avionesEnDesguaceMenorACapacidad	true
Hangar::avionesEstacionadosMenorACapacidad	true
Vuelo::identificadorUnico	true
Vuelo::pilotosDistintos	true

Segundo escenario

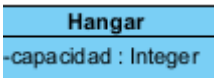
En este escenario, comprobamos que no se cumplan algunas de las invariantes creando objetos y relaciones que incumplan dichas invariantes. Podemos observar que las invariantes comprobadas devuelven como resultado ‘false’.

Class invariants

Invariant	Satisfied
Aerolinea::nombreDistinto	false
Avion::SiEstaEnHangarNoEstaEnDesguace	false
Desguace::avionEnDesguace1000Viajes	false
Desguace::avionEnDesguace2Viajes	false
Desguace::avionesEnDesguaceMenorACapacidad	true
Hangar::avionesEstacionadosMenorACapacidad	true
Vuelo::identificadorUnico	true
Vuelo::pilotosDistintos	false

Explicaciones textuales

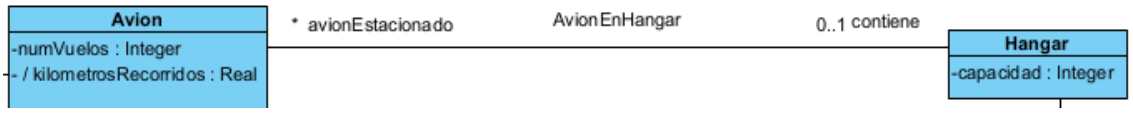
Clase Hangar



La clase Hangar cuenta con un único atributo llamado capacidad, que es un integer que almacenará la cantidad de aviones que caben en un hangar.

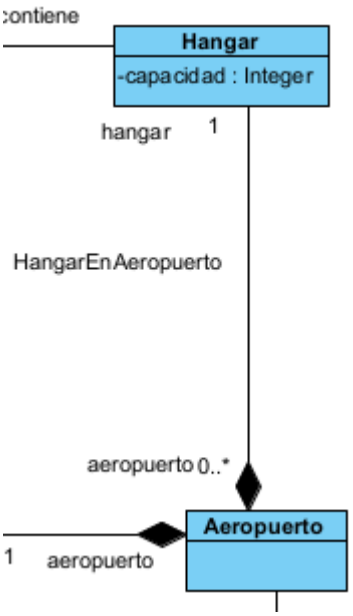
Esta cuenta con dos asociaciones:

☐ Una relación con Avión:



que nos ayudará a obtener que aviones están almacenados en el hangar, usando una multiplicidad Avión[*] <-> Hangar[0..1] debido a que podremos almacenar muchos aviones o ninguno en un o ningún hangar, ya que un avión no puede estar almacenado en más de un hangar.

☐ Una relación de composición con Aeropuerto:



Ya que los hangares serán almacenados en Aeropuerto con una multiplicidad de Aeropuerto[1] <-> Hangar[0..*] ya que puede haber muchos hangares o muchos en un único aeropuerto.

En cuanto al use, lo hemos implementado de la siguiente manera:

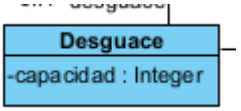
```
class Hangar
  attributes
    capacidad : Integer
end
```

y en cuanto a las relaciones:

```
composition HangarEnAeropuerto between
  Aeropuerto[1] role aeropuerto
  Hangar[0..*] role hangar
end
```

```
association AvionEnHangar between
  Avion[0..*] role avionEstacionado
  Hangar[0..1] role contiene
end
```

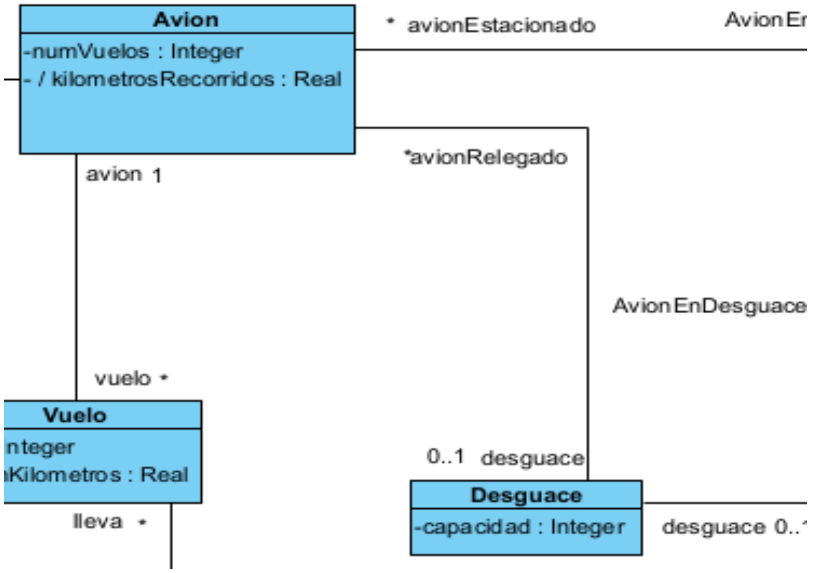
Clase Desguace



La clase desguace solo almacena el atributo de capacidad que almacena la cantidad de aviones que pueden estar en un desguace y se relaciona con Avión y con Aeropuerto mediante composición.

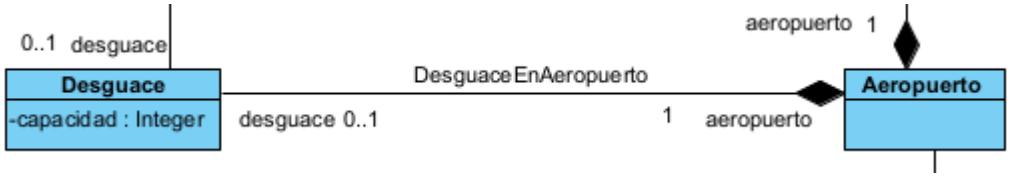
Las relaciones en las que interfiere desguace son:

☐ Una relación con Avión:



Para poder ver que aviones han sido desguazados y su información, con una multiplicidad de Avión[*] <-> Desguace[0..1], que nos permite meter muchos o ningún avión en un o ningún desguace.

☐ **Una relación de composición con Aeropuerto:**



Que muestra en qué aeropuerto se ubica el desguace con una multiplicidad de Aeropuerto[1] <-> Desguace[0..1] ya que solo puede haber un desguace a lo sumo en un aeropuerto.
En cuanto a la implementación a use:

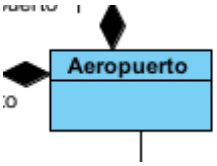
```
class Desguace
attributes
  capacidad : Integer
end
```

y sus relaciones:

```
composition DesguaceEnAeropuerto between
  Aeropuerto[1] role aeropuerto
  Desguace[0..1] role desguace
end
```

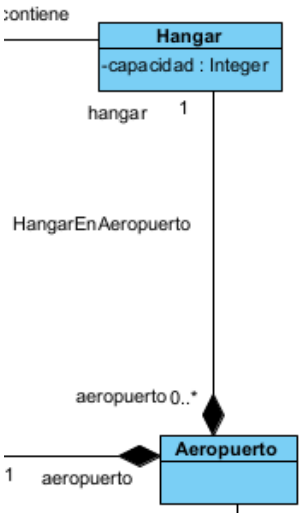
```
association AvionEnDesguace between
  Avion[0..*] role avionRelegado
  Desguace[0..1] role incluye
end
```

Clase Aeropuerto



En esta clase no tenemos ningún atributo para almacenar datos pero contamos con dos entidades que forman parte de esta por composición, Hangar y Desguace y una relación con ciudad.

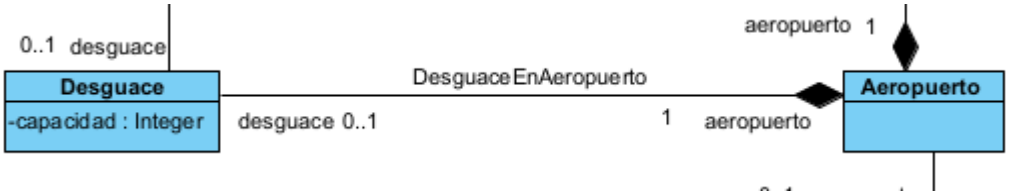
Las composiciones anteriormente dichas son:



☐ **Con Hangar:**

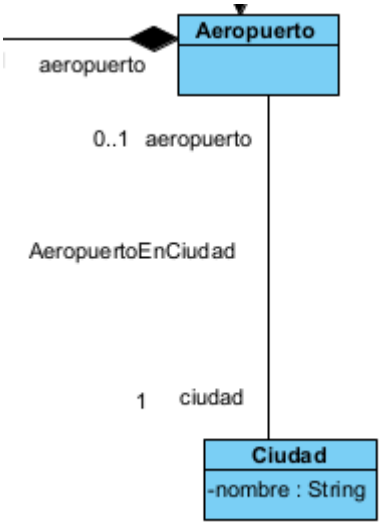
Anteriormente explicada en la clase Hangar.

☐ Con Desguace:



Anteriormente explicada en la clase Desguace.

☐ Una relación con Ciudad:



Que usaremos para almacenar en donde se encuentra el aeropuerto pudiendo tener ciudades sin aeropuerto en ellas almacenadas ya que la multiplicidad es de Aeropuerto[0..1] <-> Ciudad[1].

En cuanto al use, lo hemos implementado de la siguiente manera:

```
class Aeropuerto
end
```

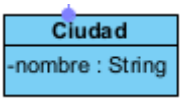
y sus relaciones:

```
composition HangarEnAeropuerto between
  Aeropuerto[1] role aeropuerto
  Hangar[0..*] role hangar
end
```

```
composition DesguaceEnAeropuerto between
  Aeropuerto[1] role aeropuerto
  Desguace[0..1] role desguace
end
```

```
association AeropuertoEnCiudad between
  Aeropuerto[0..1] role aeropuerto
  Ciudad[1] role ciudad
end
```

Clase Ciudad



Una ciudad solo almacena su nombre y se relaciona con Aeropuerto para indicar dónde se encuentra.

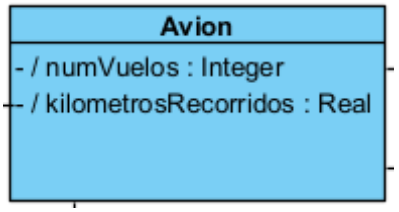
Esta solo tiene una relación con Aeropuerto anteriormente explicada en la clase Aeropuerto.

En cuanto a la implementación en use: y la relación con Aeropuerto:

```
class Ciudad
attributes
  nombre : String
end
```

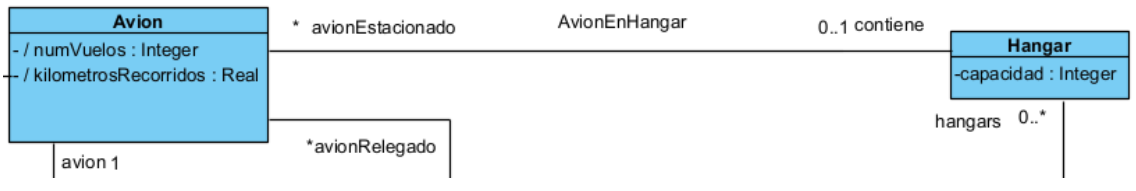
```
association AeropuertoEnCiudad between
  Aeropuerto[0..1] role aeropuerto
  Ciudad[1] role ciudad
end
```

Clase Avión



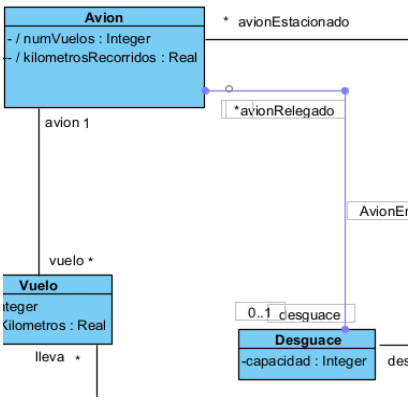
La clase Avión almacena el número de vuelos que ha realizado y los kilometrosRecorridos del Avión, ambas serán claves derivadas. Además, estas cuenta con 4 relaciones las cuales son:

- ☐ **Una relación con Hangar:**



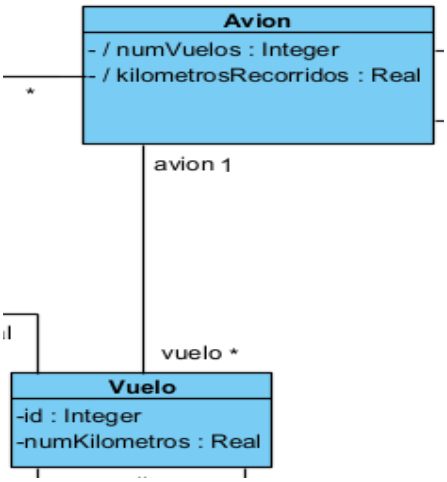
ya explicada anteriormente en la clase Hangar.

☐ **Una relación con Desguace:**



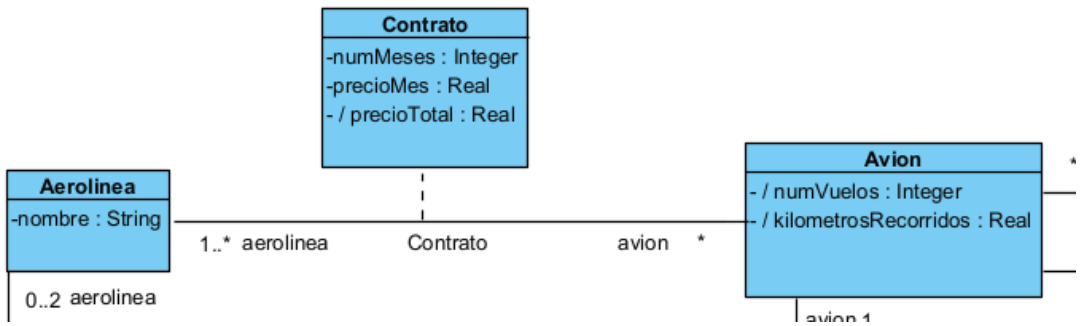
ya explicada anteriormente en la clase Desguace.

☐ **Una relación con Vuelo:**



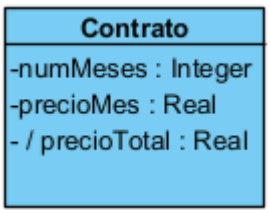
En esta asociación Avión se relaciona con Vuelo, con una multiplicidad de Avion[1] <-> Vuelo[*], es decir, un Avión que podrá hacer muchos vuelos pero un vuelo solo puede ser realizado por un Avión.

☐ **Una relación con Aerolínea mediante una clase de asociación:**



En esta relación se relaciona con Aerolínea, con una multiplicidad de Avión[*] <-> Aerolínea[1..*], que nos muestra que muchos aviones pueden tener contrato con una o más aerolíneas.

Para relacionarla, usamos la clase contrato debido a que es una relación muchos a muchos.



Esta clase cuenta con 3 atributos, numMeses que muestra el número de meses del contrato, otro atributo es precioMes, que muestra el precio por mes del contrato y un atributo derivado,precioTotal, que muestra el dinero que ganará durante todo el tiempo estipulado del contrato

En cuanto a la implementación en use:

```
class Vuelo
  attributes
    id : String
    numKilometros : Real
end
```

y sus relaciones:

```
association AvionEnHangar between
  Avion[0..*] role avionEstacionado
  Hangar[0..1] role contiene
end
```

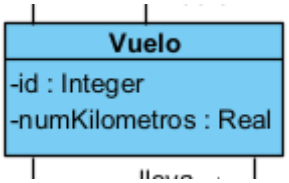
```
association AvionEnDesguace between
  Avion[0..*] role avionRelegado
  Desguace[0..1] role incluye
end
```

```
association VueloRealizado between
  Avion[1] role avion
  Vuelo[0..*] role vuelo
end
```

y la clase de asociación contrato con Aerolínea:

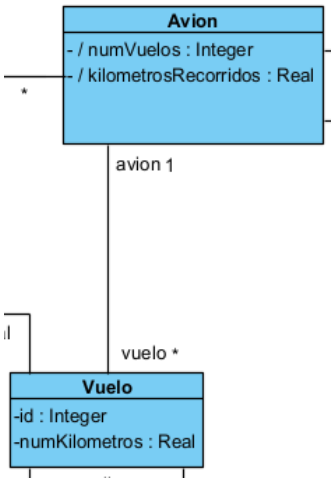
```
associationclass Contrato between
  Avion[0..*] role avion
  Aerolinea[1..*] role aerolinea
  attributes
    numMeses : Integer
    precioMes : Real
    precioTotal : Real derive : -- DERIVED
      self.numMeses * self.precioMes
end
```

Clase Vuelo



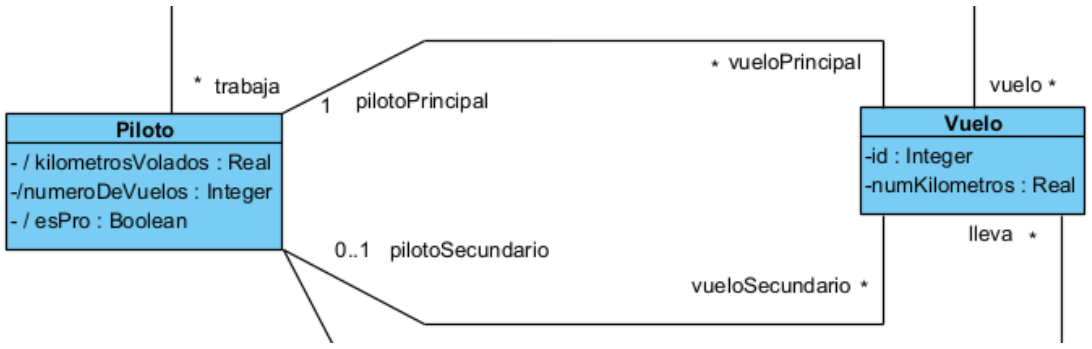
La clase vuelo cuenta con dos atributos, estos dos son “id” y “numKilometros”, El primero de ellos es un String que actúa como un identificador para el vuelo y el segundo es un real que guarda el número de kilómetros que ha recorrido el avión. Este cuenta con 4 asociaciones las cuales son:

- ☐ **Una relación con Avión:**



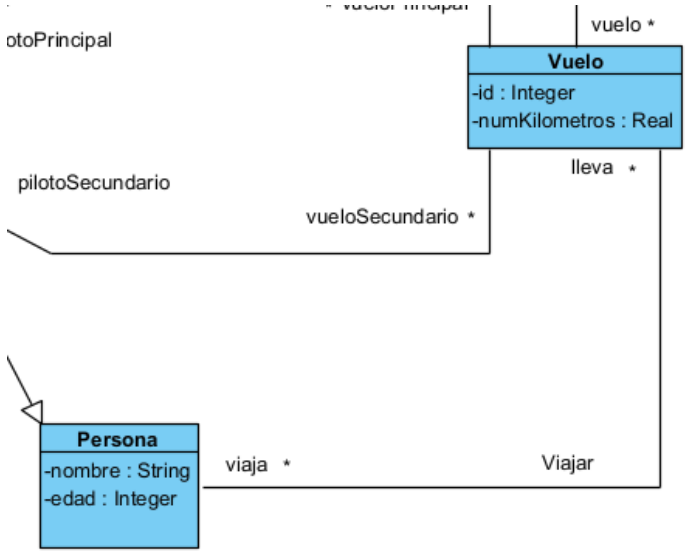
Anteriormente explicada en la clase Avión.

- ☐ **dos relaciones con la clase Piloto:**



En esta relación se relaciona con Piloto, una relación que muestra la relación de los vuelos con los pilotos primarios con una multiplicidad de Avión[*]<->Piloto[1], ya que un piloto primario puede haber asistido a muchos vuelos pero cada vuelo solo puede tener un piloto primario. La otra relación que muestra la relación con los pilotos secundarios con una multiplicidad de Avión[*]<->Piloto[0..1] ya que puede haber a lo sumo un piloto secundario en cada vuelo, pero un mismo piloto secundario ha podido asistir a muchos vuelos.

- ☐ **una relación con la clase Persona:**



En esta relación se relaciona con Persona, con una multiplicidad de Vuelo[*]<->Persona[*], que muestra que muchas personas pueden estar en un vuelo y que una persona puede asistir a muchos vuelos.

En cuanto a la implementación en use:

```
class Vuelo
  attributes
    id : String
    numKilometros : Real
end
```

y sus relaciones:

```
association VueloRealizado between
  Avion[1] role avion
  Vuelo[0..*] role vuelo
end
```

```
association Volar between
  Persona[0..*] role pasajero
  Vuelo[0..*] role lleva
end
```

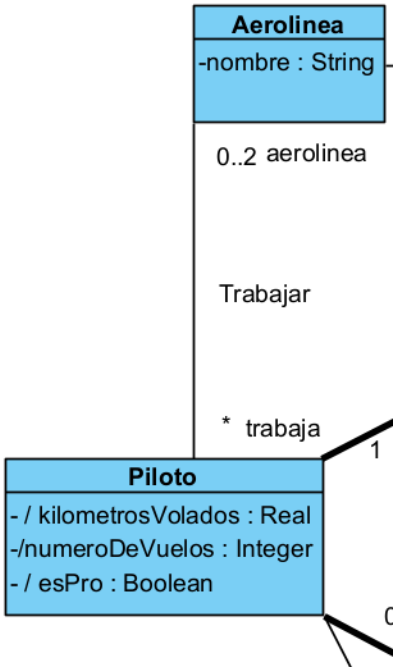
```
association PilotoPrincipal between
  Piloto[1] role pilotoPrincipal
  Vuelo[0..*] role vueloPrincipal
end
```

```
association PilotoSecundario between
  Piloto[0..1] role pilotoSecundario
  Vuelo[0..*] role vueloSecundario
end
```

Clase Aerolínea

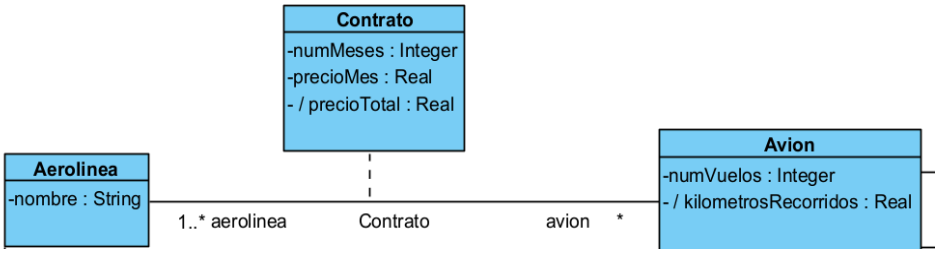
La clase aerolínea cuenta con un atributo denominado “nombre” que es de clase String que nos aporta el nombre de la aerolínea. Esta clase cuenta con una única relación:

- ☐ **Relacion “Trabajar” con Piloto:**



Aquí tenemos una cardinalidad Piloto[0..*]<-->Aerolínea[0..2] donde una aerolínea puede contar con un número indefinido(pudiendo ser 0) de pilotos, mientras que estos pueden trabajar en un máximo de 2 Aerolíneas distintas(pudiendo no trabajar en ninguna)

- ☐ **Relacion “Contrato” con Avión:**

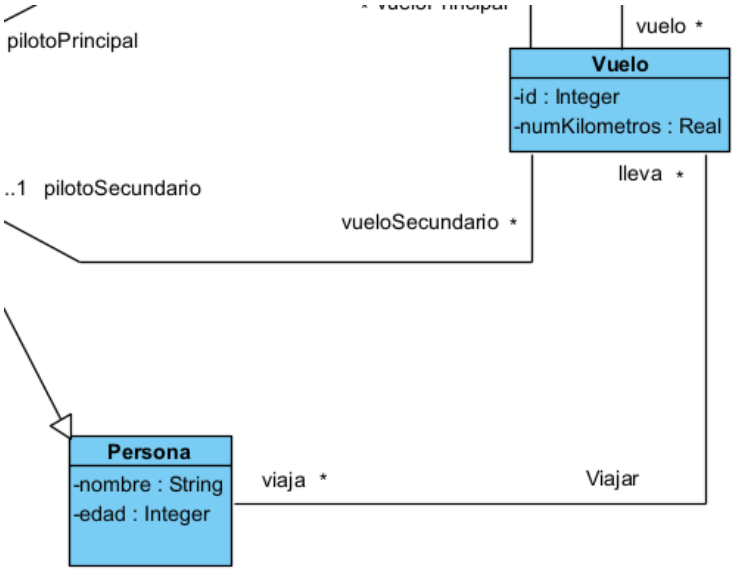


Relación explicada anteriormente

Clase Persona

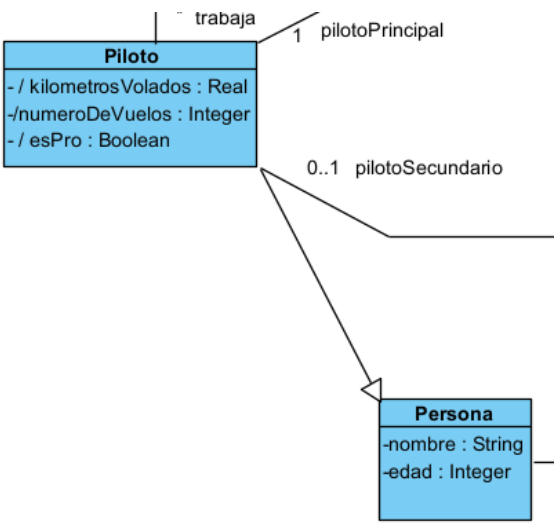
La clase Persona cuenta con dos atributos: nombre (de tipo String) y edad (de tipo Integer). Esta clase cuenta con dos asociaciones:

- ☐ **Una relación con la clase Vuelo:**



Ya explicada anteriormente

- ☐ **Una relación de generalización con la clase Piloto:**

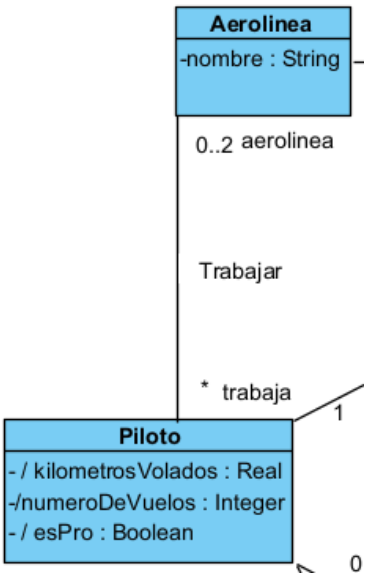


ya que piloto es un tipo de persona

Clase Piloto

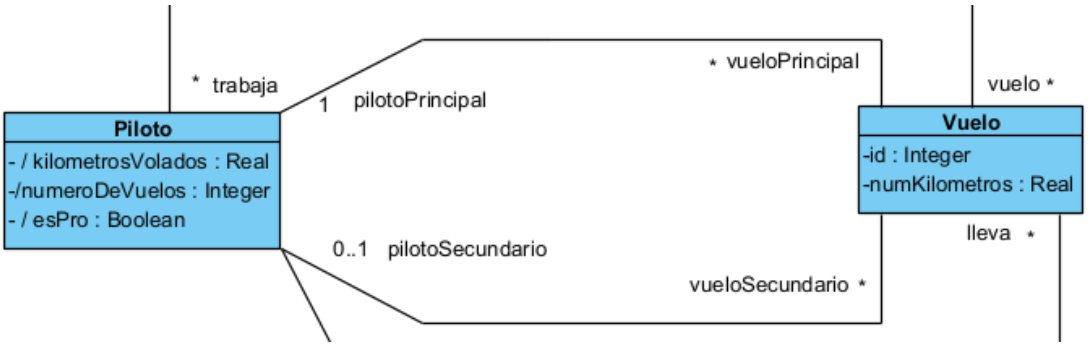
La clase piloto cuenta con tres atributos. El primero es un real llamado “KilometrosVolados” que te dice cuántos kilómetros ha recorrido el piloto. El segundo de ellos es un Integer llamado “NumeroDeVuelos” que te dice la cantidad de vuelos que ha recorrido el piloto. Por último, hay un atributo booleano denominado “esPro” que devuelve true or false dependiendo de si el piloto es pro o no. Cuenta con las siguientes relaciones:

- ☐ **una relación con la clase Aerolínea:**



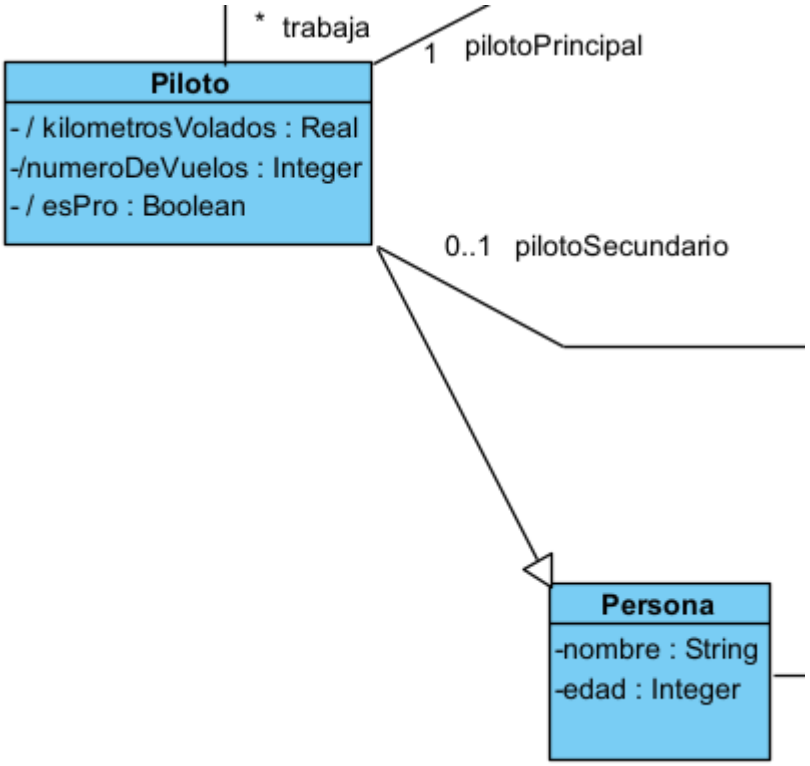
Relación explicada anteriormente

- ☐ **Dos relaciones con la clase Avión:**



Relación explicada anteriormente

☐ **Una relación con la clase Persona:**



Relación explicada anteriormente.