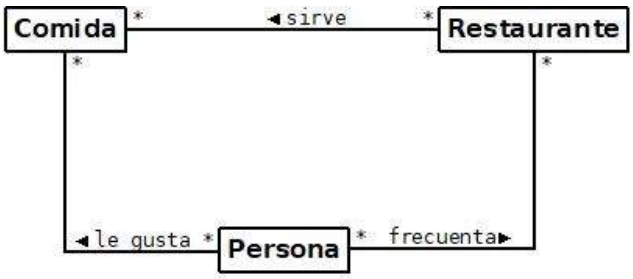
Práctico de Programación 4 – clase 3

Práctico 2, Ejercicio 3:

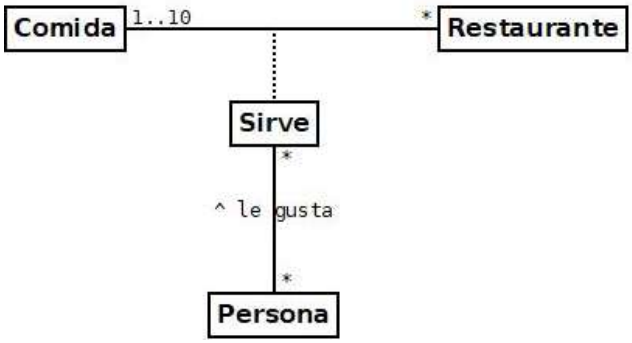
En una primera aproximación consideramos la siguiente solución, que cumple con la primera parte de la letra.



Notar que NO cumple con las restricciones especificadas en la segunda parte de la letra del ejercicio, en particular con las siguientes:

- *Un restaurante no sirve más de 10 comidas.* En la solución propuesta, la multiplicidad * (que es una abreviación de 0..*) del lado de Comida significa que un Restaurante puede no servir comidas y no hay un límite superior en la cantidad de comidas que sirve.
- *A una persona no le gusta una comida por sí sola sino como la sirven en determinados restaurantes, aunque puede no gustarle ninguna.* En la solución propuesta, a una Persona le gusta una Comida, pero esa Comida es servida por muchos Restaurantes, por lo tanto, no está representada la predilección por una Comida de un Restaurante específico.

Para cumplir con estas dos restricciones consideramos la siguiente solución:



Notar que:

- El 1..10 del lado de Comida establece cantidades mínima y máxima de instancias de la clase Comida, con las que se puede relacionar una instancia de Restaurante.
- El tipo asociativo Sirve representa una Comida servida en un Restaurante. No es lo mismo la pizza del Bar Toto que la pizza de la Rotisería Nona.

Tener en cuenta que una asociación R entre dos clases A y B se define como un subconjunto del producto cartesiano de A y B, es decir, $R \subseteq A \times B$. Entonces, una instancia válida del modelo anterior podría ser la siguiente:

- Comida = {pizza, chivito, milanesa}
- Restaurante = {Bar Toto, Rotisería Nona}
- Persona = {Pepe}
- Sirve = {(pizza, Bar Toto), (pizza, Rotisería Nona)}
- LeGusta = {(Pepe, (pizza, Bar Toto))}

Recordar que el modelo de dominio representado por el diagrama puede verse como una plantilla que permite generar todas las instancias válidas de la realidad que se está modelando. Es importante no confundir el nombre de la clase Comida (se escribe en singular, plantilla para generar cualquier comida) con el conjunto de comidas de una instancia particular (en el ejemplo de más arriba, el conjunto de comidas que tiene los elementos pizza, chivito y milanesa). Notar que para simplificar se hace abuso del lenguaje, pero en el primer caso Comida refiere a una clase, mientras que en el segundo caso refiere al conjunto de comidas de una instancia particular del modelo. Una clase NO es un conjunto, en particular, no es el conjunto de todas las posibles comidas ni un subconjunto del mismo.

Sugerencia: Probar con otras instancias (conjuntos de Comida, Restaurante, Persona, Sirve) y verificar su validez en relación al modelo expresado en el diagrama.

Notar que:

- Sirve es también una clase, por lo tanto, podría tener atributos, por ejemplo, un precio (la pizza del Bar Toto es más barata que la pizza de la Rotisería Nona).
- Estamos asumiendo que si a una Persona le gusta una instancia de Sirve (un par (Comida, Restaurante)), entonces frecuenta el Restaurante. Si en la realidad esto no fuera necesariamente cierto, entonces además debería existir la relación Persona frecuenta Restaurante, permitiendo que la Persona frecuente Restaurantes donde no necesariamente Sirven comidas que le gusten.

Respecto a las otras dos restricciones:

- *Una persona frecuenta varios restaurantes*. Está modelada por el * (límite superior no acotado) del lado de Sirve en la relación le gusta.
- *Una comida servida por un restaurante puede no gustarle a ninguna persona*. Está modelada por el * (límite inferior 0) del lado de Persona en la relación le gusta.

Práctico 2, Ejercicio 4:

a)
Lo primero a observar es que la multiplicidad de la asociación vende del lado de Producto tiene * como máximo, mientras que la letra establece que un Vendedor vende un único Producto. Esto puede solucionarse cambiando la multiplicidad 1..* por 1.

Una instancia válida del modelo, en principio podría ser la siguiente. En particular, notar que cumple con los mínimos de las multiplicidades de las asociaciones, que en todos los casos es 1:

- Empresa = {e1, e2, e3}
- Vendedor = {v1, v2, v3}
- Producto = {p1, p2, p3}
- Trabaja = {(e1, v1), (e2, v2), (e3, v3)}
- Produce = {(e1, p1), (e2, p2), (e3, p3)}
- Vende = {(p1, v2), (p2, v3), (p3, v1)}

Notar que el vendedor v2 vende el producto p1 que es producido por la empresa e1, pero el vendedor v2 trabaja en la empresa e2 (no es la misma que produce el producto que vende, lo que contradice la letra del problema).

Lo anterior podría solucionarse con una restricción de la siguiente forma:

> *Si una instancia v de Vendedor está asociada a una instancia de Producto a través de vende, que a su vez está asociada a una instancia e1 de Empresa a través de produce, y v está asociada con una instancia e2 de Empresa a través de trabaja, entonces e1 = e2.*

Notar que en la redacción de la restricción tratamos de ser lo más precisos posibles, con las limitaciones del lenguaje natural. En particular:

- Hacemos referencia textual a los nombres de las clases y asociaciones del modelo.
- Introducimos algunos identificadores específicos (v, e1, e2) para facilitar las expresiones.

Se puede escribir algo más sencillo, por ejemplo, sin nombrar las asociaciones, dado que en este caso no hay más de una asociación entre cualquier par de clases:

> *Si una instancia v de Vendedor está asociada a una instancia de Producto, que a su vez está asociada a una instancia e1 de Empresa, y v está asociada con una instancia e2 de Empresa, entonces e1 = e2.*

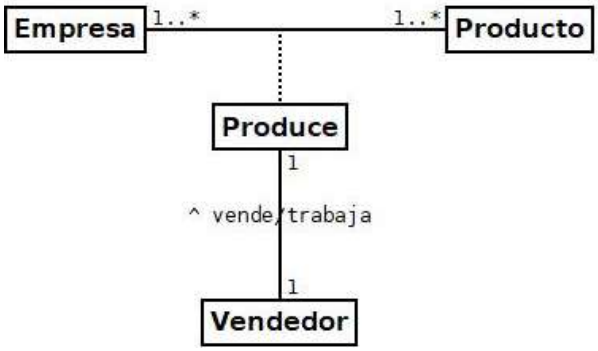
Se debe tener cuidado con la simplificación en la escritura de las restricciones. En particular lo siguiente NO es válido:

> *Un vendedor vende un producto de la empresa para la que trabaja.*

Notar que no hace referencia a ningún componente del modelo (clase, asociación) y se parece más a una parte de la letra del problema que a una restricción del modelo que se está construyendo.

b)

Una alternativa podría ser la siguiente:



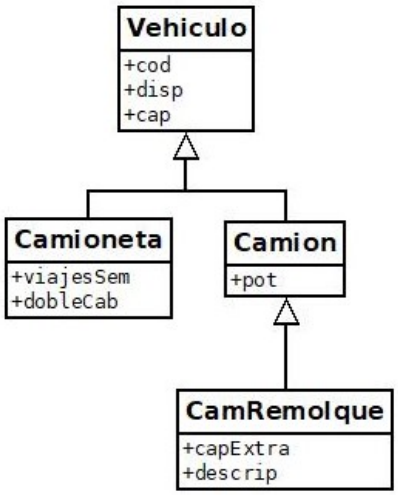
Notar que el nombre de la asociación entre Produce y Vendedor es vende/trabaja, para reflejar el hecho de que si el Vendedor v está asociado al par Produce (e,p), entonces v vende p y v trabaja en e.

c)

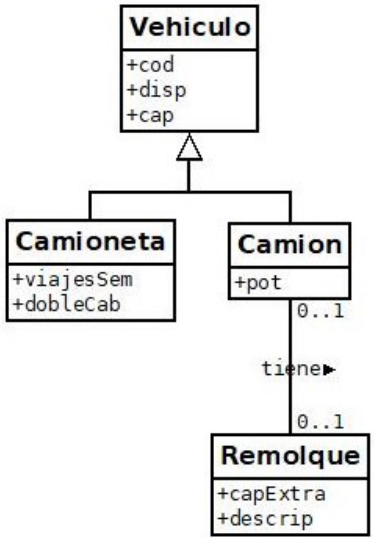
En la solución propuesta en la parte b), si un Producto p no sale a la venta, estará asociado de todos modos con las Empresas e que lo producen (al menos con una), pero no tendrá un Vendedor asociado. Por lo tanto, la multiplicidad de la asociación vende/trabaja del lado de Vendedor deberá ser 0..1. En el modelo original del ejercicio, la multiplicidad de la asociación vende del lado de Vendedor deberá ser 0..1.

Práctico 2, Ejercicio 6:

Se consideran las dos siguientes alternativas:



(a)

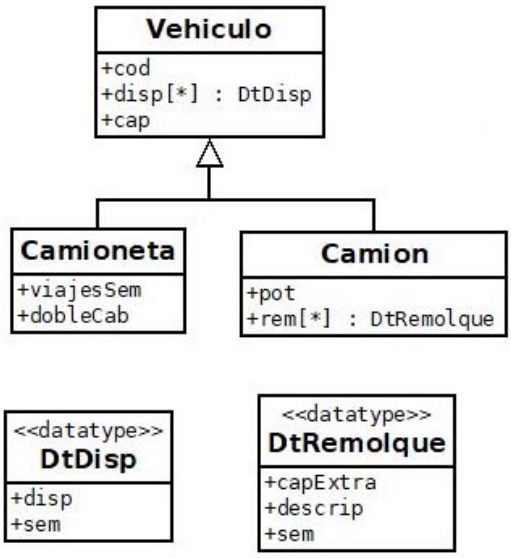


(b)

Observar que en la alternativa (a) se asume que un camión con remolque es efectivamente un vehículo, y en principio siempre será de la clase CamRemolque. Si un camión en un momento tiene remolque y en otro momento no tiene, debe dejar de existir la instancia de CamRemolque y crearse la de Camion (con los mismos datos, en particular, el identificador). Notar que esto puede ser discutible, el objeto está prácticamente cambiando de clase.

La alternativa (b) considera que los remolques pueden ser intercambiables. Por lo tanto, una instancia de Camion puede estar en un momento asociada a ninguna instancia de Remolque y en otro momento puede estar asociada a uno.

Ante la pregunta de cual es la mejor opción, la respuesta es que depende del contexto. Si no se necesita modelar información histórica, la alternativa (a) es la mejor. En este caso, el estado del sistema refleja el estado actual de la flota. Si fuera necesario registrar información histórica, por ejemplo, todas las semanas del año, se propone la alternativa (c), ver en la siguiente página. En este caso, la información que depende de la semana del año (disponibilidad del vehículo y asociación al remolque) se replica para cada una de las semanas. Notar que el atributo disp en la clase Vehiculo es multivaluado (se indica con [*]), uno para cada semana. Análogamente, el Camion tiene un atributo que indica en qué semanas tuvo Remolque, junto con los datos del mismo.


Vehiculo
+cod
+disp[*] : DtDisp
+cap
Camioneta
+viajesSem
+dobleCab
Camion
+pot
+rem[*] : DtRemolque
<<datatype>>
DtDisp
+disp
+sem
<<datatype>>
DtRemolque
+capExtra
+descrip
+sem

(c)