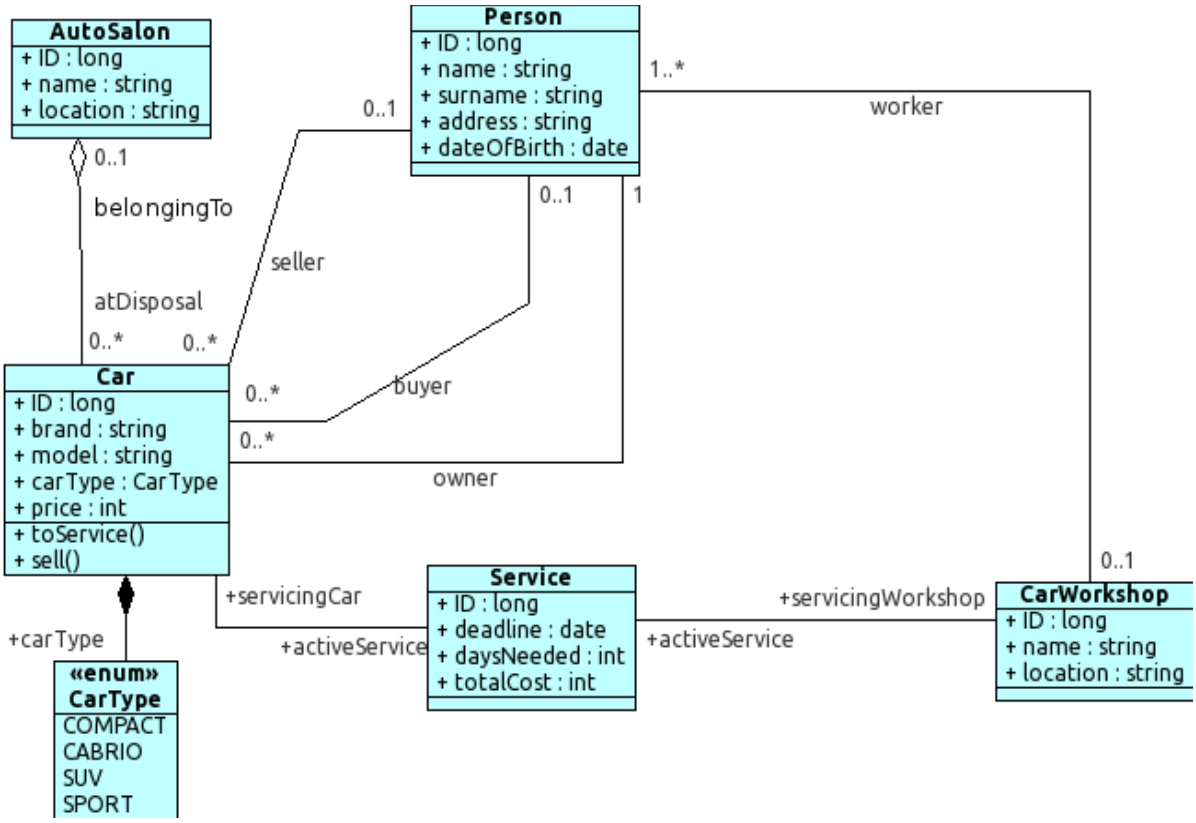
Take the following class diagram as a reference. It models some of the interactions in an *auto salon*. An *auto salon* has several cars at disposal to be sold, as it connects sellers and buyers once they agree on the price. A *person* can be a seller and/or a buyer. *Cars* can also go to service, this means that if they need reparation, they will be sent to a *car workshop* that is owned by the *auto salon*.



The UML diagram does not model some characteristics of the domain. Try to define them by using *OCL*:

[1] When the car is sold (by calling the *sell()* operation), the number of cars at disposal within the *Autosalon* must be lower than the total number before executing the operation.
[2] A new law is introduced in the legislation, that states that only people that is born before 1.1.1982 can own a car of type *SPORT*
[3] after the *toService()* operation has been performed, an active service for the car must be existing, as well, the the same service must be associated to a car workshop

1.
```
context Car::sell():void post:
self.belongingTo.atDisposal()->size()<
self.belongingTo.atDisposal()->size()@pre
```

2.
```
context Car inv:
def limitDate:String = "1.1.1982"
self.carType::SPORT implies (self.owner.dateOfBirth <
limitDate.oclAsType(Date))
```

Note: there is no "Date" class in OCL, but depends on the implementation

Now the tricky aspect (apart from notation, as we have seen during the lecture):

```
context Person inv:
def limitDate:String = "1.1.1982"
self.dateOfBirth  <  limitDate.oclAsType(Date)  implies
self.carType::SPORT
```

From the notation (in the example shown there was a wrong **...implies Car::carType::SPORT),** this is correct, but this is a kind of different constraint. We are reporting that all the instances of type Person that are born before 1.1.1982 have a car of type SPORT! (but if you read [2], here the point is that only people born before 1.1.1982 can own a car of SPORT (so the constraint is on the type), they still can own other types of cars if they want! (with this constraint, we do not allow this)

3.
```
context Car::toService() post:
self.activeService <> oclVoid AND
self.activeService.servicingWorkshop <> oclVoid
```

Note, from the UML diagram we can derive that cardinality in all the relations activeService and servicingWorkshop is 1-1. If you considered them as collections, you will check then for **activeService->size()>0** and **activeService->select(m | id > 0)->servicingWorkshop-> notEmpty()**