**sig** Integer {}

**sig** Strings {}

**sig** Date {}

**sig** Time {}

**sig** Coordinate{}

**enum** DriverStatus {Busy, Available}

**enum** RideStatus{Annulled, Assigned, Completed, NotAssigned}

**abstract** **sig** User {}

**sig** Guest **extends** User {}

**sig** Admin **extends** User{

name: **one** Strings,

surname: **one** Strings,

email: **one** Strings,

password: **one** Strings

}

**abstract** **sig** RegUser **extends** User{

name: **one** Strings,

surname: **one** Strings,

password: **one** Strings,

birth: **one** Date,

email: **one** Strings,

telephone: **one** Strings

}

**sig** Customer **extends** RegUser{}

**sig** TaxiDriver **extends** RegUser{

status: **one** DriverStatus,

taxi: **one** Taxi

}

**sig** Taxi {

code: **one** Integer,

position: **one** Coordinate

}

**sig** TaxiQueue {

hasTaxi: set Taxi

}

**sig** TaxiZone {

hasTaxiQueue: **one** TaxiQueue

}

**one** **sig** Map {

hasZone: some TaxiZone

}

**one** **sig** System {

map: **one** Map,

users: set User,

taxiRide: set TaxiRide

}

**abstract** **sig** TaxiRide {

origin: **one** Coordinate,

destination: **lone** Coordinate,

date: **one** Date,

time: **one** Time,

waitingTime: **lone** Integer,

taxi: **lone** Taxi,

rideStatus: **one** RideStatus,

hasCustomer: **one** Customer

}

**sig** Request **extends** TaxiRide {

}

**sig** Reservation **extends** TaxiRide {

}

//\*\*\*\***Fact**s

//No duplicate users

noDuplicateUsers{

**no** **disj** u1,u2: RegUser| (u1.email = u2.email) **or** (u1.telephone = u2.telephone)

**no** **disj** a1,a2:Admin | (a1.email = a2.email)

**no** **disj** a1:Admin, u1:RegUser | (a1.email = u1.email)

}

//Every taxi has always exactly one driver

taxiOneDriver{

**no** **disj** d1,d2:TaxiDriver | d1.taxi = d2.taxi

#TaxiDriver = # Taxi

TaxiDriver <: taxi **in** TaxiDriver **one**->Taxi

}

//Every taxi queue has always exactly one taxi zone

**fact** queueOneZone{

TaxiZone <: hasTaxiQueue **in** TaxiZone **one** -> TaxiQueue

}

//Every taxi can be in only one queue

**fact** taxiOneQueue{

**all** t1:Taxi | **no** **disj** q1,q2:TaxiQueue | (t1 **in** q1.hasTaxi **and** t1 **in** q2.hasTaxi)

**all** t2:Taxi | **lone** q3:TaxiQueue | t2 **in** q3.hasTaxi

}

//the map has all the taxi zones

**fact** taxiZoneInMap{

TaxiZone **in** Map.hasZone

}

//No more than 1 "Assigned" taxiRide for customer

**fact** RideLimit{

**no** **disj** r1,r2:TaxiRide | (r1.rideStatus = r2.rideStatus) **and** (r2.rideStatus = Assigned) **and**

(r1.hasCustomer = r2.hasCustomer)

**no** **disj** r1,r2:Request | (r1.rideStatus = r2.rideStatus) **and** (r2.rideStatus = NotAssigned) **and**

(r1.hasCustomer = r2.hasCustomer)

**no** **disj** r1,r2:Request | (r1.rideStatus = Assigned) **and** (r2.rideStatus = NotAssigned) **and**

(r1.hasCustomer = r2.hasCustomer)

**no** **disj** r1:Request, r2:Reservation| (r1.rideStatus = NotAssigned) **and** (r2.rideStatus = Assigned) **and**

(r1.hasCustomer = r2.hasCustomer)

}

//taxi (or taxi driver) has no more than 1 taxiRide with assigned status

**fact** driverOneRide{

**no** **disj** r1,r2:TaxiRide | (r1.taxi = r2.taxi) **and** (r1.rideStatus = Assigned) **and** (r2.rideStatus = r1.rideStatus)

}

//no taxi paired with a taxi ride with "not assigned" status

**fact** noTaxiNotAssigned{

**all** r1:TaxiRide | (r1.rideStatus = NotAssigned) **implies** (r1.taxi = **none**)

}

//Assigned and completed rides must be bound to a taxi

**fact** TaxiRideStatus{

**all** r1:TaxiRide | (r1.rideStatus != NotAssigned **and** r1.rideStatus!= Annulled) **implies** (#r1.taxi=1)

}

//systems must have the reference to all users and rides

**fact** systemUserRide{

**all** u1:User | u1 **in** System.users

**all** r1:TaxiRide | r1 **in** System.taxiRide

}

// annulled rides must not be linked to any taxi

**fact** AnnulledNoTaxi{

//**no** r:TaxiRide | r.rideStatus = Annulled **and** r.taxi != **none**

**all** r:TaxiRide | r.rideStatus = Annulled **implies** r.taxi = **none**

}

//busy taxi must **not** be **in** a queue

**fact** taxiBusyQueue{

**all** d1:TaxiDriver | ((d1.status = Busy) **implies** (**no** q1:TaxiQueue | d1.taxi **in** q1.hasTaxi))

// **no** t:Taxi | taxi.t.status **in** Busy **and** #hasTaxi.t >0

//**all** t:Taxi| taxi.t.status **in** Busy **implies** #hasTaxi.t=0

}

//available taxi must be **in** a queue

**fact** taxiAvailableQueue{

**all** d1:TaxiDriver | ((d1.status = Available) **implies** (some q1:TaxiQueue | d1.taxi **in** q1.hasTaxi))

//**all** t:Taxi | taxi.t.status **in** Available **implies** #hasTaxi.t >0

}

//Origin cannot be equal to destination

**fact** originDifferentDestination{

**no** r1:TaxiRide | (r1.origin = r1.destination)

}

// assigned taxi implies busy taxi driver

**fact** assignedEqualBusy{

**no** t:Taxi | taxi.t.rideStatus **in** Assigned **and** taxi.t.status=Available

}

// available taxidriver implies noassigned taxiRide

**fact** availableEqualnotAssigned{

**all** t:Taxi,d:TaxiDriver | (d.taxi = t **and** d.status = Available) **implies**(**no** r:TaxiRide | r.taxi = t **and** r.rideStatus = Assigned)

}

// all this cardinality equivalences must be satisfied

**assert** numbersEquivalence{

#Taxi = #TaxiDriver

#TaxiZone = #TaxiQueue

#System.users = # User

#System.taxiRide = #TaxiRide

# (status.Available).taxi = #TaxiQueue.hasTaxi // the number of available taxi must be equal to the number of taxi in all the queues

}

**check** numbersEquivalence

//busy taxi drivers may not be assigned to a ride, because of the possibility of standard customers

**assert** driversStandardCustomer{

**all** s1,s2:System | addBusyWithoutRide[s1,s2] **implies** !(status.Busy **in** (TaxiDriver <: taxi).(TaxiRide.(TaxiRide <:taxi)))

}

//**check** driversStandardCustomer

// all available taxi belong to a taxi zone

**assert** allAvailableInOneZone{

**all** t:Taxi | ((TaxiDriver <: taxi).t).status = Available **implies** **one** z:TaxiZone | t **in** z.hasTaxiQueue.hasTaxi

}

**check** allAvailableInOneZone

// no taxi without drivers

**assert** noTaxiWithoutDrivers{

**all** t:Taxi | **one** d:TaxiDriver | d.taxi=t

}

//**check** noTaxiWithoutDrivers

// no drivers without taxi

**assert** noDriversWithoutTaxi{

**all** d:TaxiDriver | **one** t:Taxi | d.taxi=t

}

**check** noDriversWithoutTaxi

// no ride without customers

**assert** noRideWithoutCustomer{

**all** r:TaxiRide | **one** c:Customer| r.hasCustomer = c

}

**check** noRideWithoutCustomer

//Other commands

//Add an assigned ride

**pred** addAssignedRide(s1,s2:System){

**one** r1:TaxiRide | r1.rideStatus = Assigned **and** s2.taxiRide=s1.taxiRide + r1

}

//Add an annulled ride

**pred** addAnnulledRide(s1,s2:System){

**one** r1:TaxiRide | r1.rideStatus = Annulled **and** s2.taxiRide=s1.taxiRide + r1

}

//Add a completed ride

**pred** addCompletedRide(s1,s2:System){

**one** r1:TaxiRide | r1.rideStatus = Completed **and** s2.taxiRide=s1.taxiRide + r1

}

//Add 2 taxi drivers

**pred** add2Driver (s1,s2:System, d1,d2:TaxiDriver){

s2.users= s1.users + d1 + d2

}

//Add 2 customers

**pred** add2Customer(s1,s2:System,c1,c2:Customer){

s2.users=s1.users + c1+c2

}

//Add 2 reservations

**pred** add2Reservation(s1,s2:System, res1,res2:Reservation){

s2.taxiRide=s1.taxiRide + res1 + res2

}

//Add 2 requests

**pred** add2Request(s1,s2:System, req1,req2:Request){

s2.taxiRide=s1.taxiRide + req1 + req2

}

//Add 1 busy and 1 available taxi driver

**pred** atleast1busy1available{

some d:TaxiDriver | d.status = Busy

some d:TaxiDriver | d.status = Available

}