# Joe Restaurant

Module Software Engineering 2, DT228 Bsc in Computer Science, Dublin Institute of Technology

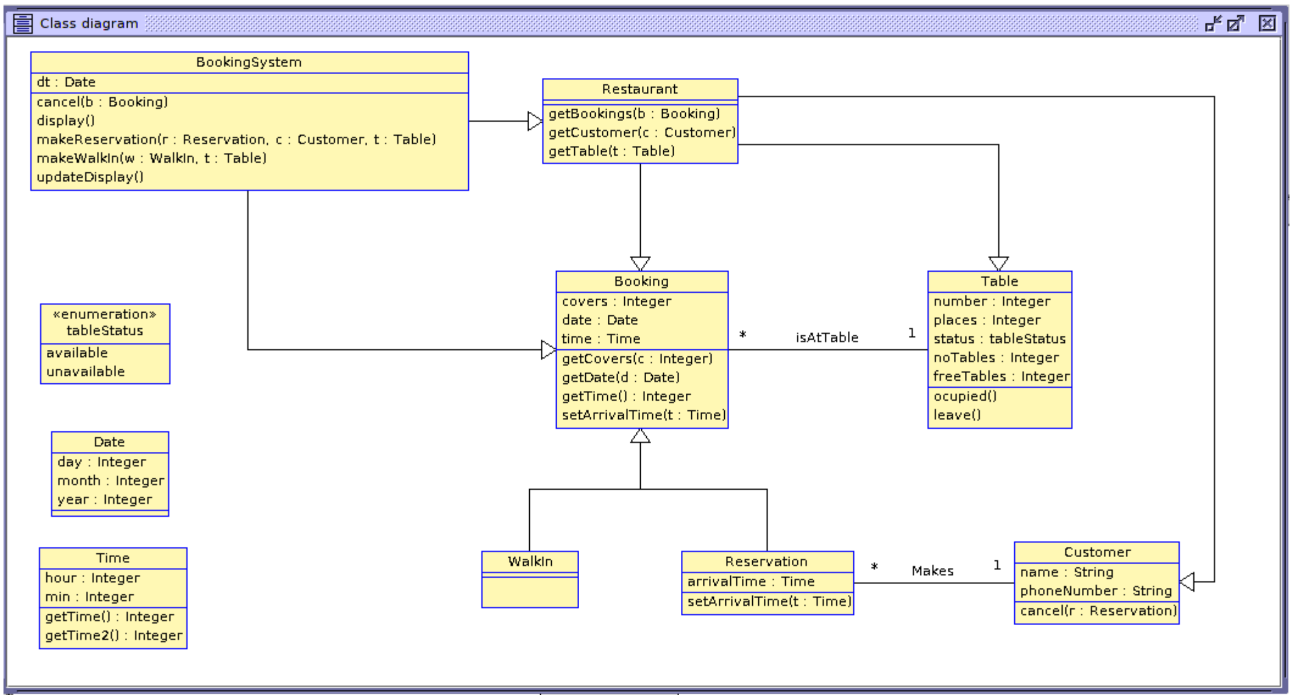
**Iosif Bogdan Dobos (C16735789)**

| | |
|---|---|
| Student Name: | Iosif Bogdan Dobos |
| Student Number: | C16735789 |
| Lecturer's Name: | Richard Lawlor |
| Module: | Software Engineering 2 |
| Submission Date: | 5th May 2018 |

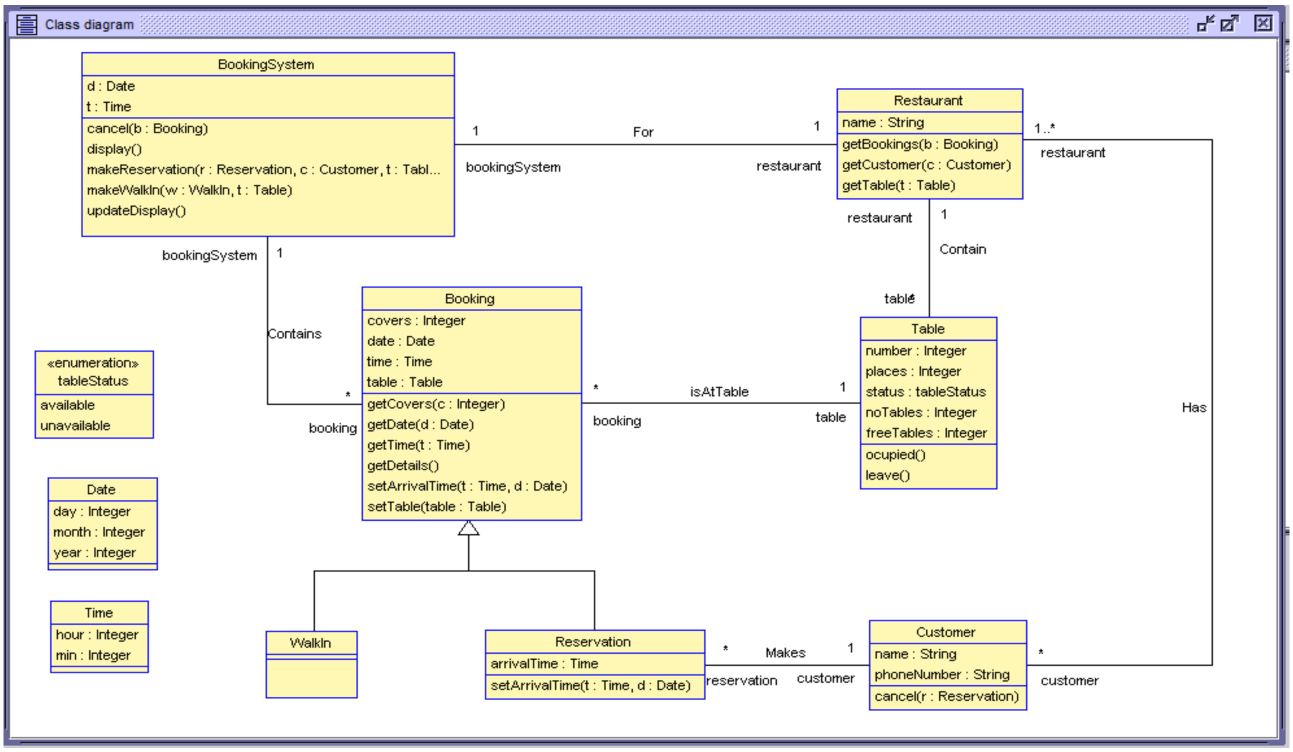
## Introduction

For This assignment I choose to do the restaurant case study. The system that I have developed is mainly used to support day-to-day operations of a restaurant by improving the process of making reservations and allocation tables to customers.

## Class Diagram

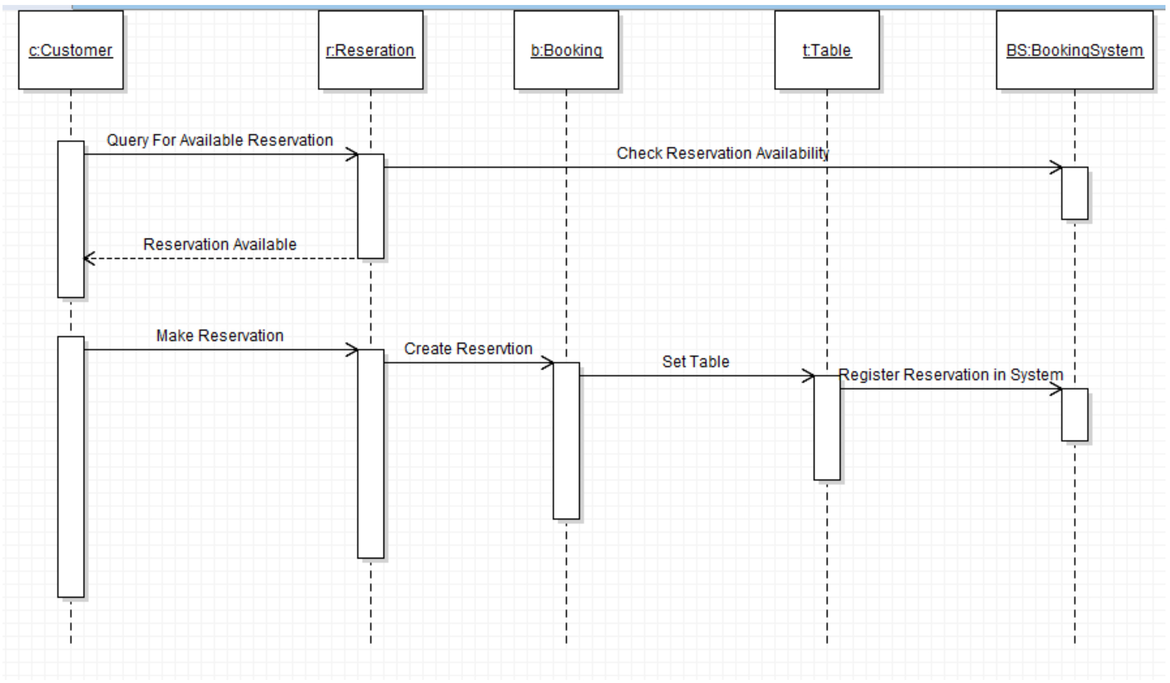


Class diagram after implementing OCL

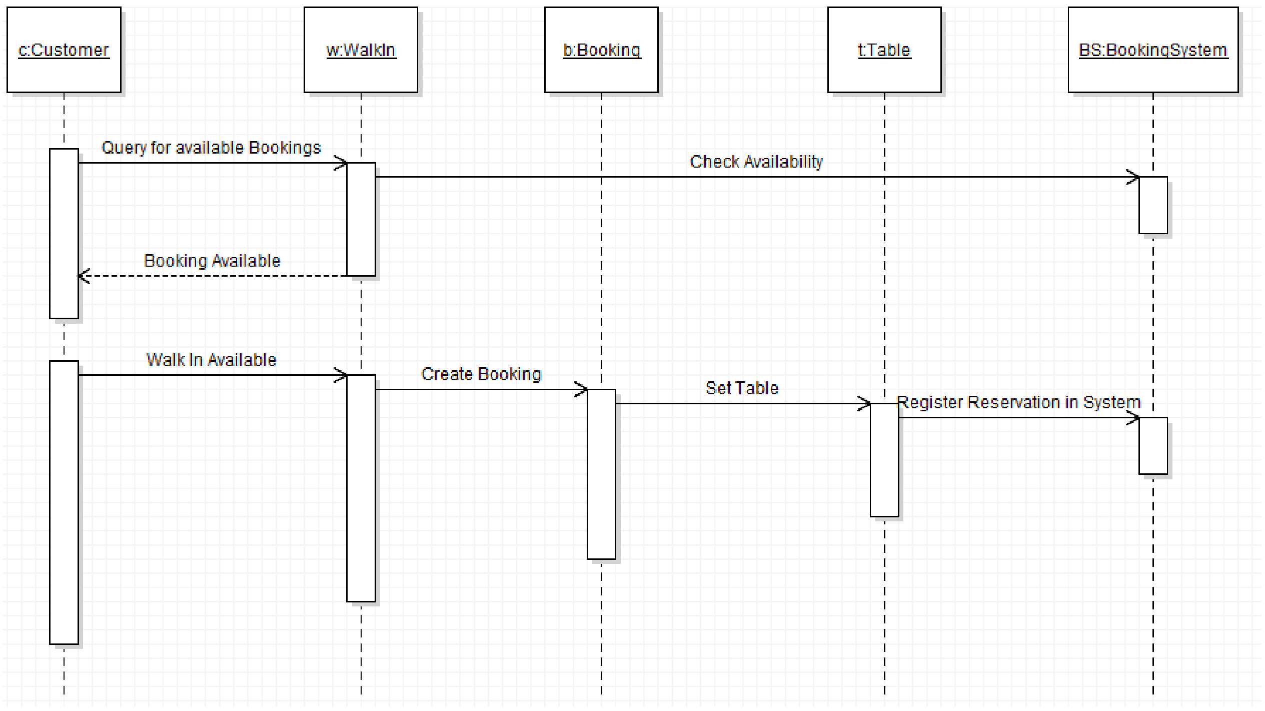


## Sequence Diagram

Customer makes reservation



Customer walks into restaurant

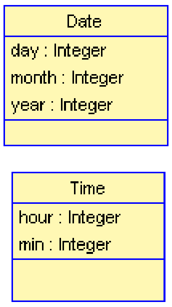


## Use Implementation

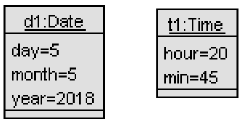
In the Restaurant case study, customers can make reservation to the restaurant for dinner. In order to do the reservation customers must provide some information like the name to make the reservation for and a phone number to be contacted. Before making a reservation tables from the restaurant needs to be available to make the reservation possible. Also, we need to check customers who walks to make reservation for them as well in order to avoid customers to have dinner at the same table and time. All this reservation that are made are stored in the Booking System.

## Date and Time information required

Because in the restaurant model, when making a reservation the time and date for is crucial and as in USE there is no built-in date and time data types, I have created myself two different table where I stored the date and time for the specific day and I can use it as a data type for implementing the booking system. I called this two table Date and Time and I assign to it some attributes as shown in the diagram bellow.

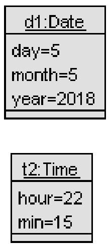


When a reservation needs to be make, a Date and Time object has to be created and when the arrival time needs to be set, it will be assign to that respective object. For example, I need to make a reservation on my system on 05/05/2018 at 20:0. The time and date will be made as follows:



After I stored this values in my model, I can use this date for setting arrival time and storing date and time for a reservation.

Another example can be a reservation on 06/05/2018 at 22:15 --->

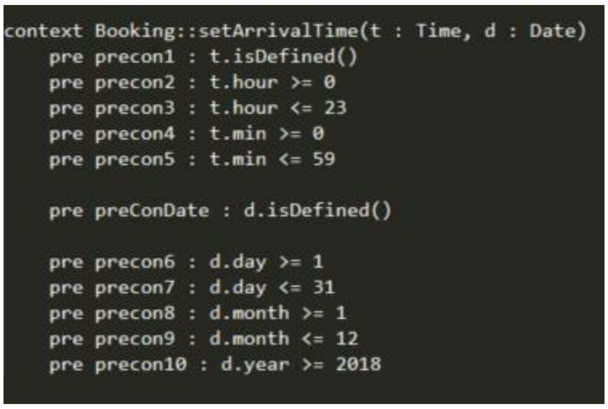


Because people can make lots of errors, they can accidentally insert wrong values. Values like hour 25 should not be correct because there is no hour 25 in a day, or the number of minutes shouldn't be bigger than 60, day should be between 1 and 31, month between 1 and 12 and year shouldn't be greater than current year as we do not want to make reservations in past.

For all this mistakes that can occur, I have create some pre conditions every time when an object of this time is created. Let's take an example.

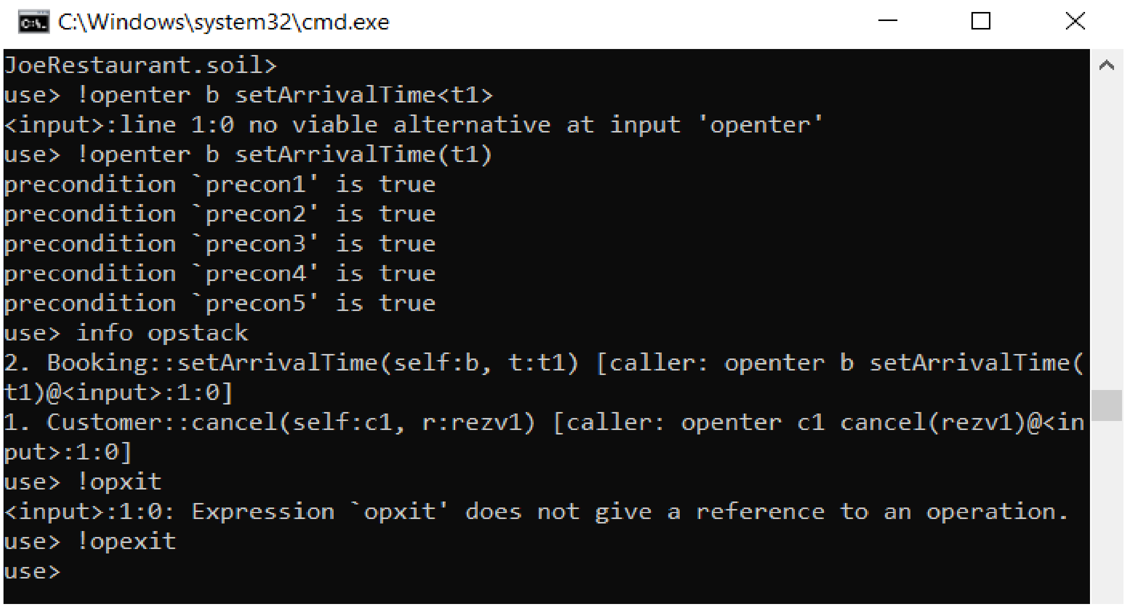
We create the object d and t: Date and Time in the first example from above. We want to use the information when we make a booking. When assign the d and t values to arrivalTime from booking, I have to use !operner and !opexit to check the preconditions so there will be no mistakes in my model.

The preconditions are the following:



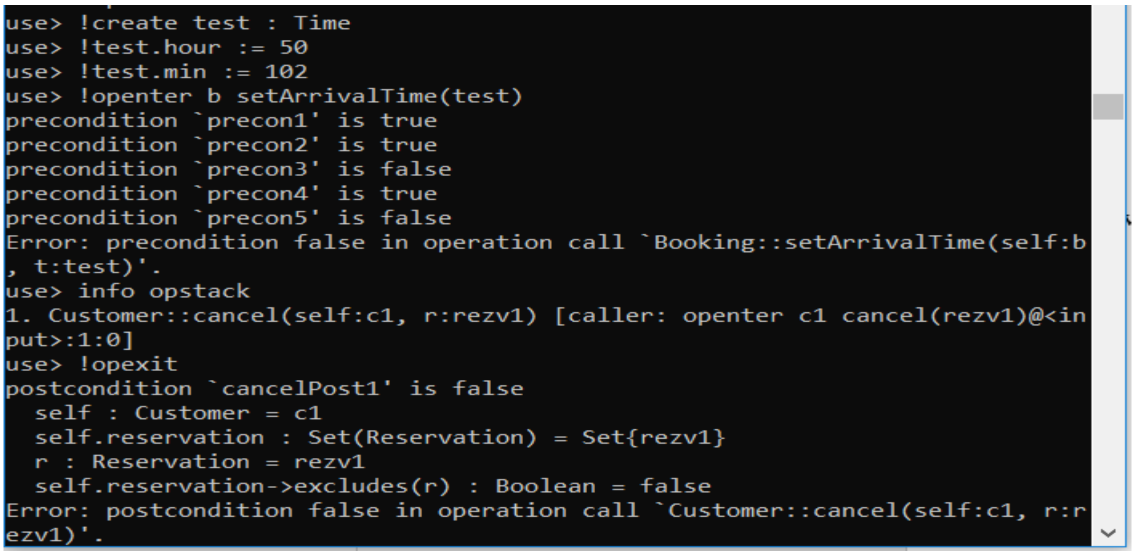
```
context Booking::setArrivalTime(t : Time, d : Date)
    pre precon1 : t.isDefined()
    pre precon2 : t.hour >= 0
    pre precon3 : t.hour <= 23
    pre precon4 : t.min >= 0
    pre precon5 : t.min <= 59

    pre preConDate : d.isDefined()

    pre precon6 : d.day >= 1
    pre precon7 : d.day <= 31
    pre precon8 : d.month >= 1
    pre precon9 : d.month <= 12
    pre precon10 : d.year >= 2018
```

The way needs to be check the pre-condition are like:



```
C:\Windows\system32\cmd.exe
JoeRestaurant.soil>
use> !openter b setArrivalTime<t1>
<input>:line 1:0 no viable alternative at input 'openter'
use> !openter b setArrivalTime(t1)
precondition `precon1' is true
precondition `precon2' is true
precondition `precon3' is true
precondition `precon4' is true
precondition `precon5' is true
use> info opstack
2. Booking::setArrivalTime(self:b, t:t1) [caller: openter b setArrivalTime(
t1)@<input>:1:0]
1. Customer::cancel(self:c1, r:rezv1) [caller: openter c1 cancel(rezv1)@<in
put>:1:0]
use> !opxit
<input>:1:0: Expression `opxit' does not give a reference to an operation.
use> !opexit
use>
```

All conditions are true here, because the values are between the limits they are supposed to be. I have created another object that will make the pre-condition to be false:

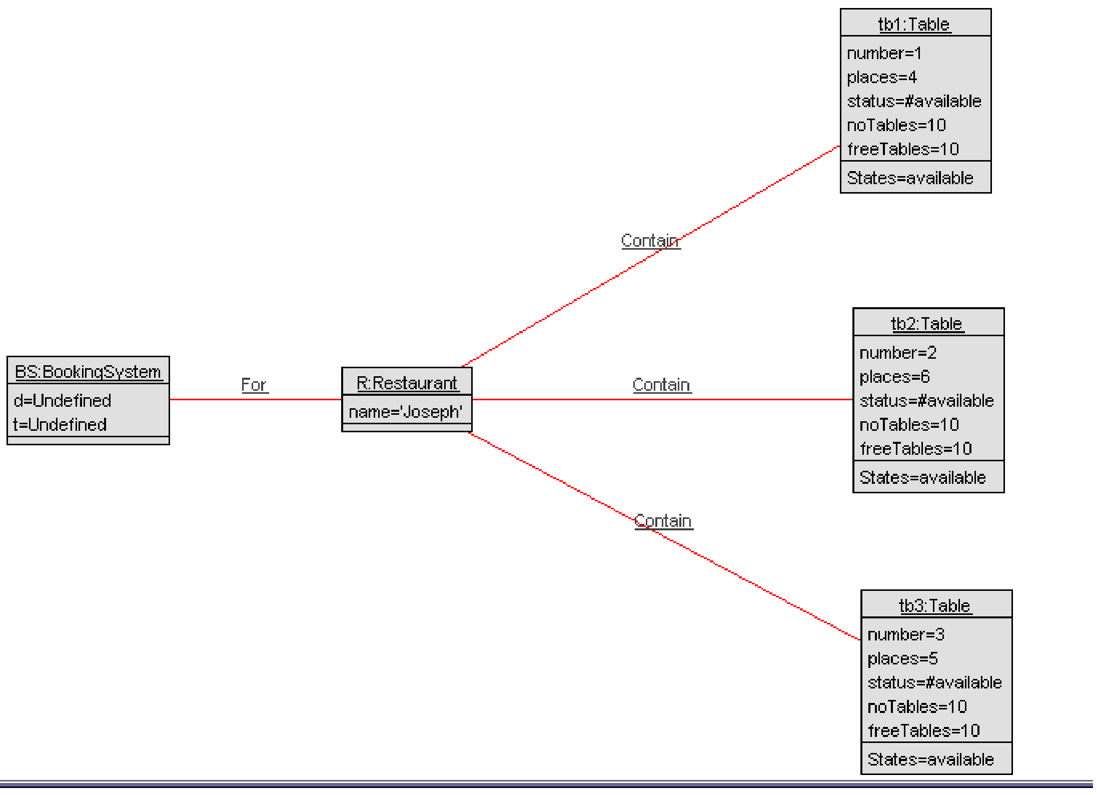


```
use> !create test : Time
use> !test.hour := 50
use> !test.min := 102
use> !openter b setArrivalTime(test)
precondition `precon1' is true
precondition `precon2' is true
precondition `precon3' is false
precondition `precon4' is true
precondition `precon5' is false
Error: precondition false in operation call `Booking::setArrivalTime(self:b
, t:test)'.
use> info opstack
1. Customer::cancel(self:c1, r:rezv1) [caller: openter c1 cancel(rezv1)@<in
put>:1:0]
use> !opexit
postcondition `cancelPost1' is false
  self : Customer = c1
  self.reservation : Set(Reservation) = Set{rezv1}
  r : Reservation = rezv1
  self.reservation->excludes(r) : Boolean = false
Error: postcondition false in operation call `Customer::cancel(self:c1, r:r
ezv1)'.
```

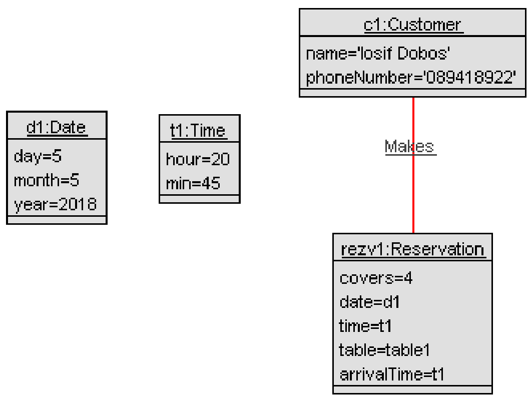
As its shows in the diagram above the precon3 and precon5 are false due to the fact that the time is not in 24-hour range and the minutes are not in the 60 range which make the preconditions to be false.

## Make reservation

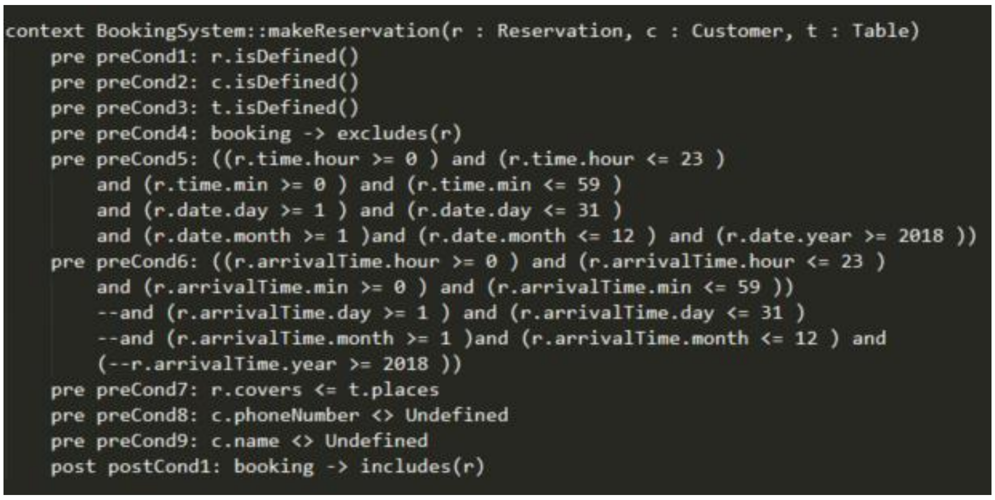
I have created a restaurant object with 10 tables on it, and on the diagram bellow is shown the tables assign to booking system.



After I did this, I am going to create a reservation rezv1 for a customer c1 on the Date and Time I have created above d and t.

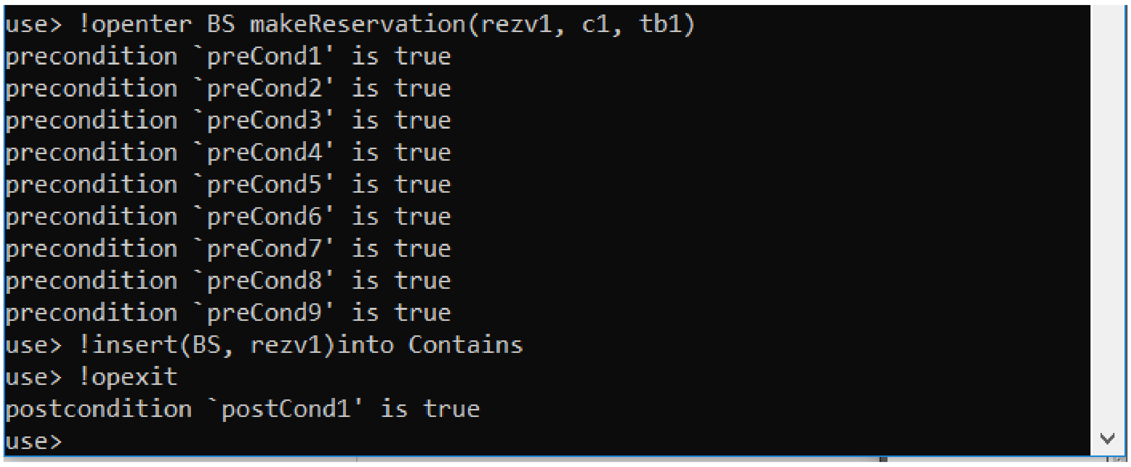


Next I have checked the precondition for make reservation connected to the Booking System. The precondition are as follows:



```
context BookingSystem::makeReservation(r : Reservation, c : Customer, t : Table)
    pre preCond1: r.isDefined()
    pre preCond2: c.isDefined()
    pre preCond3: t.isDefined()
    pre preCond4: booking -> excludes(r)
    pre preCond5: ((r.time.hour >= 0 ) and (r.time.hour <= 23 )
        and (r.time.min >= 0 ) and (r.time.min <= 59 )
        and (r.date.day >= 1 ) and (r.date.day <= 31 )
        and (r.date.month >= 1 )and (r.date.month <= 12 ) and (r.date.year >= 2018 ))
    pre preCond6: ((r.arrivalTime.hour >= 0 ) and (r.arrivalTime.hour <= 23 )
        and (r.arrivalTime.min >= 0 ) and (r.arrivalTime.min <= 59 ))
        --and (r.arrivalTime.day >= 1 ) and (r.arrivalTime.day <= 31 )
        --and (r.arrivalTime.month >= 1 )and (r.arrivalTime.month <= 12 ) and
        (--r.arrivalTime.year >= 2018 ))
    pre preCond7: r.covers <= t.places
    pre preCond8: c.phoneNumber <> Undefined
    pre preCond9: c.name <> Undefined
    post postCond1: booking -> includes(r)
```

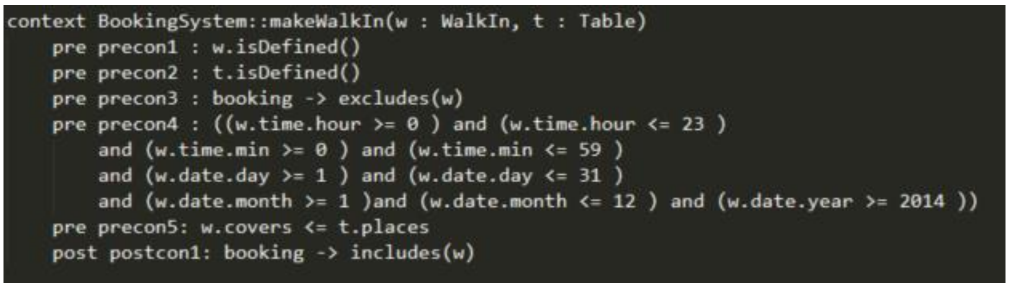
Checking the preconditions with !openter and !opexit as follow:



```
use> !openter BS makeReservation(rezv1, c1, tb1)
precondition `preCond1' is true
precondition `preCond2' is true
precondition `preCond3' is true
precondition `preCond4' is true
precondition `preCond5' is true
precondition `preCond6' is true
precondition `preCond7' is true
precondition `preCond8' is true
precondition `preCond9' is true
use> !insert(BS, rezv1)into Contains
use> !opexit
postcondition `postCond1' is true
use>
```

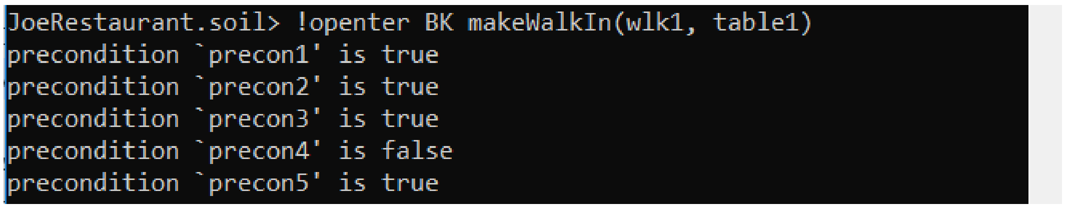
Register Walk In

In a walk-in situation, there is no need for a name or a phone number of the customer to make the reservation. If there are available tables in the restaurant then they can be allocated to any people. In order to register this in my restaurant model, I have to create a Walk-in object, which inherits from booking, and register it in the system with openter and opexit.

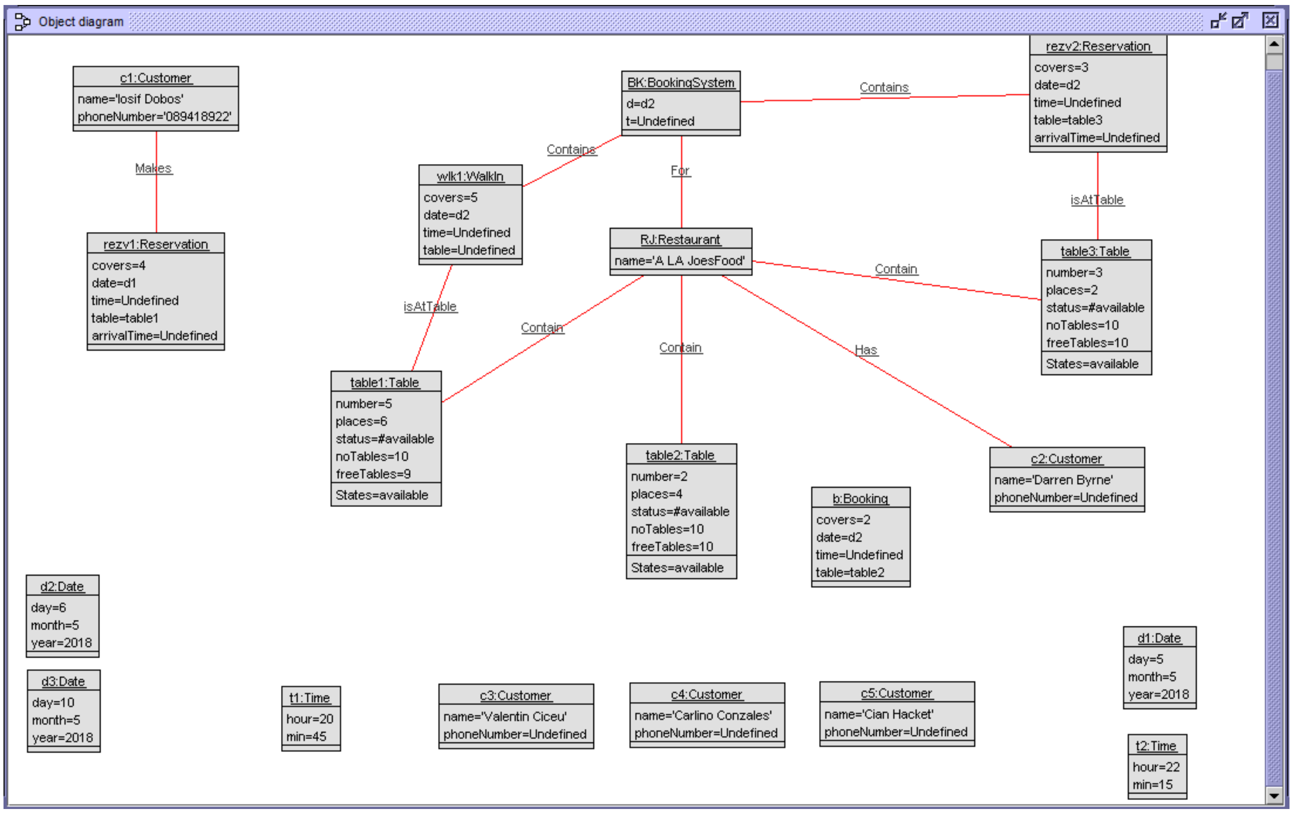


```
context BookingSystem::makeWalkIn(w : WalkIn, t : Table)
    pre precon1 : w.isDefined()
    pre precon2 : t.isDefined()
    pre precon3 : booking -> excludes(w)
    pre precon4 : ((w.time.hour >= 0 ) and (w.time.hour <= 23 )
        and (w.time.min >= 0 ) and (w.time.min <= 59 )
        and (w.date.day >= 1 ) and (w.date.day <= 31 )
        and (w.date.month >= 1 )and (w.date.month <= 12 ) and (w.date.year >= 2014 ))
    pre precon5: w.covers <= t.places
    post postcon1: booking -> includes(w)
```

After executing openter and opexit I got the walk in registred successful.

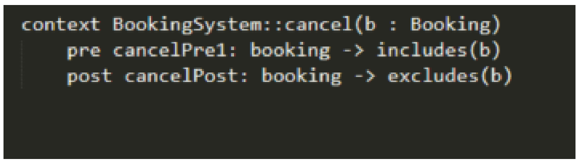


```
JoeRestaurant.soil> !openter BK makeWalkIn(wlk1, table1)
precondition `precon1' is true
precondition `precon2' is true
precondition `precon3' is true
precondition `precon4' is false
precondition `precon5' is true
```



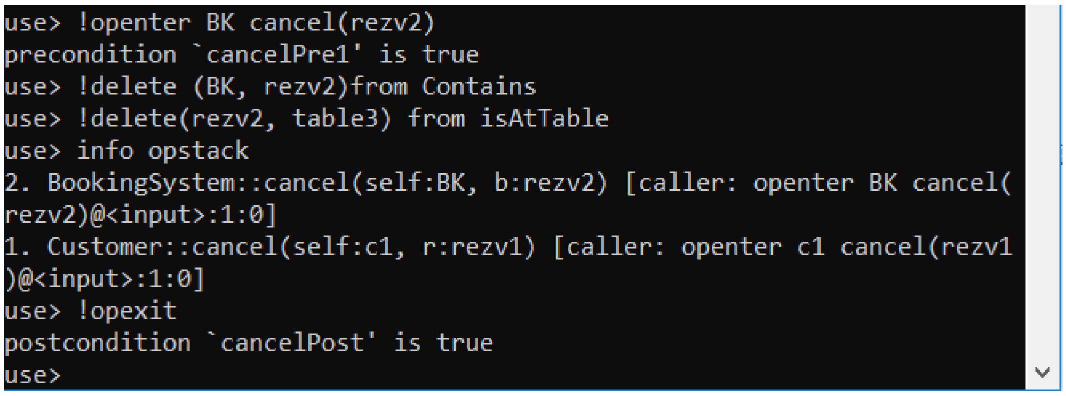
## Cancel Booking

I can also cancel Bookings from my system. Just for showing that this is working, I will cancel the rez1 reservation, and I will create that again.

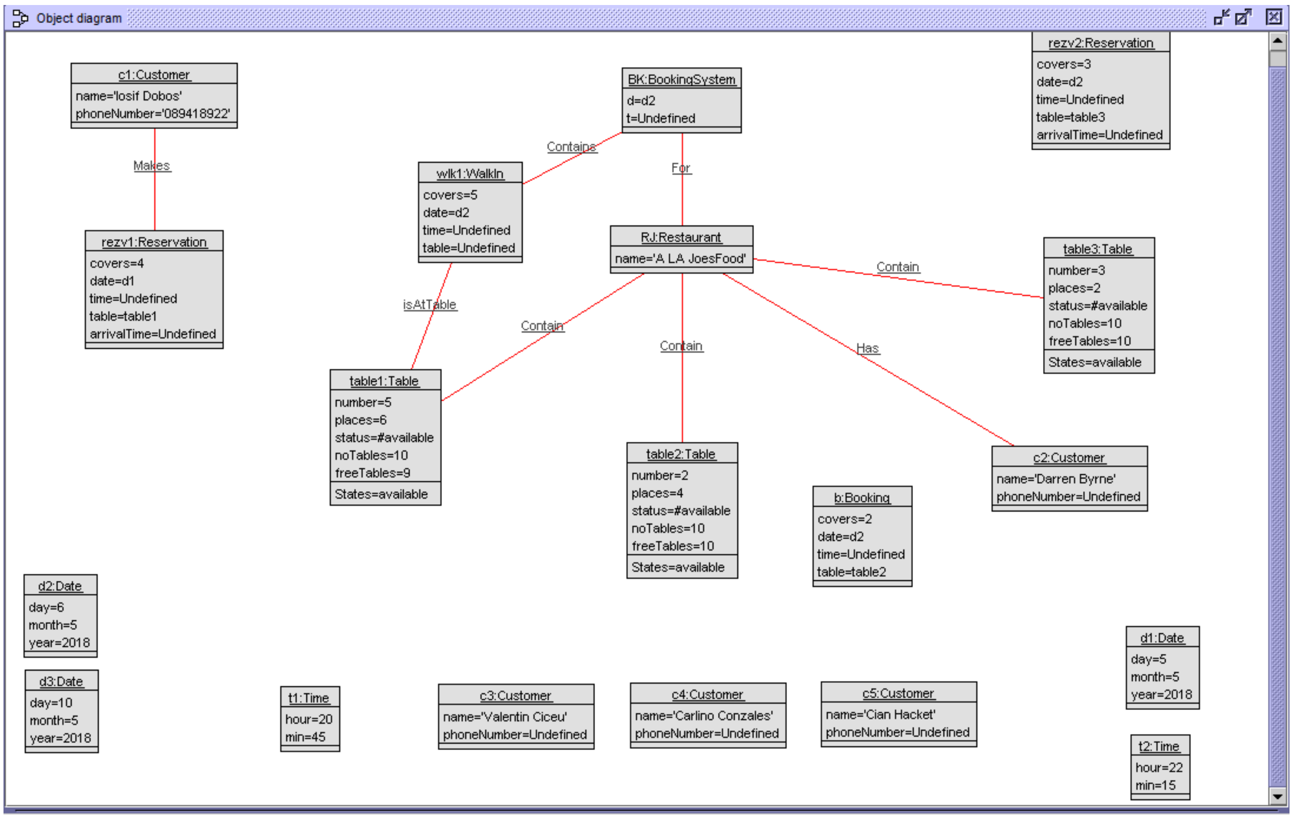


```
context BookingSystem::cancel(b : Booking)
    pre cancelPre1: booking -> includes(b)
    post cancelPost: booking -> excludes(b)
```

Testing cancel booking in use:



```
use> !openter BK cancel(rezv2)
precondition `cancelPre1' is true
use> !delete (BK, rezv2)from Contains
use> !delete(rezv2, table3) from isAtTable
use> info opstack
2. BookingSystem::cancel(self:BK, b:rezv2) [caller: openter BK cancel(
rezv2)@<input>:1:0]
1. Customer::cancel(self:c1, r:rezv1) [caller: openter c1 cancel(rezv1
)@<input>:1:0]
use> !opexit
postcondition `cancelPost' is true
use>
```

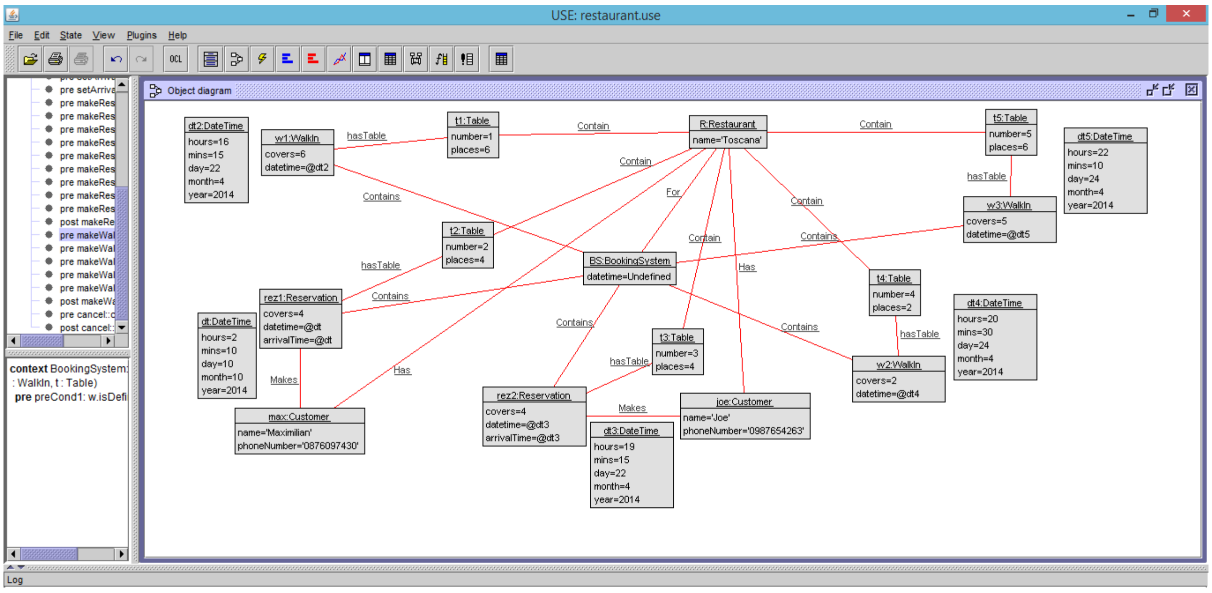


**Populate model with more Reservations**

I will now create some more reservations, one for each table so I can make some more invariants after this.

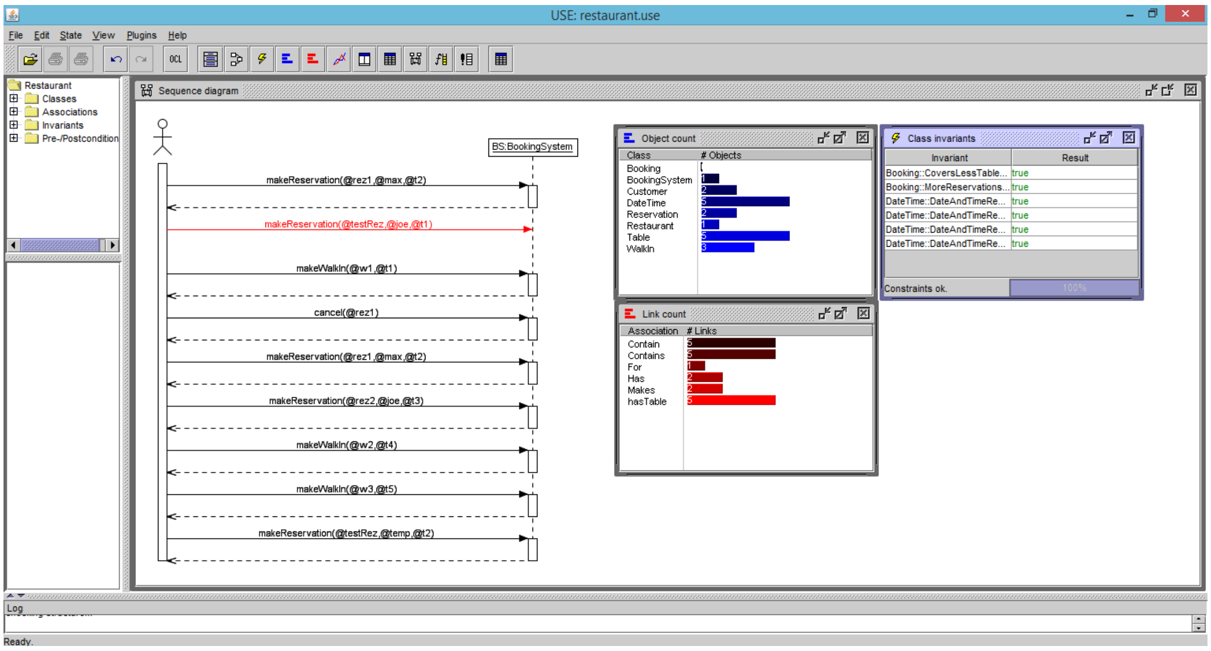
Each time when I will add a reservation in the system, I will check it with openter and opexit.

The next



```
context BookingSystem:
: WalkIn, t : Table)
 pre preCond1: w.isDefi
```

image is showing the sequence diagram view, object count view, link count view and class invariant view. The sequence diagram can be extended pressing right click > Show all Commands in USE.


USE: restaurant.use
File Edit State View Plugins Help
Restaurant
Classes
Associations
Invariants
Pre-/Postcondition
Sequence diagram
BS:BookingSystem
makeReservation(@rez1,@max,@t2)
makeReservation(@testRez,@joe,@t1)
makeWalkIn(@w1,@t1)
cancel(@rez1)
makeReservation(@rez1,@max,@t2)
makeReservation(@rez2,@joe,@t3)
makeWalkIn(@w2,@t4)
makeWalkIn(@w3,@t5)
makeReservation(@testRez,@temp,@t2)
Log
Ready.

Object count

| Class | # Objects |
|---|---|
| Booking | |
| BookingSystem | 1 |
| Customer | 2 |
| DateTime | 5 |
| Reservation | 2 |
| Restaurant | 1 |
| Table | 5 |
| WalkIn | 3 |

Link count

| Association | # Links |
|---|---|
| Contain | 5 |
| Contains | 5 |
| For | 1 |
| Has | 2 |
| Makes | 2 |
| hasTable | 5 |

Class invariants

| Invariant | Result |
|---|---|
| Booking::CoversLessTable... | true |
| Booking::MoreReservations... | true |
| DateTime::DateAndTimeRe... | true |
| DateTime::DateAndTimeRe... | true |
| DateTime::DateAndTimeRe... | true |
| DateTime::DateAndTimeRe... | true |

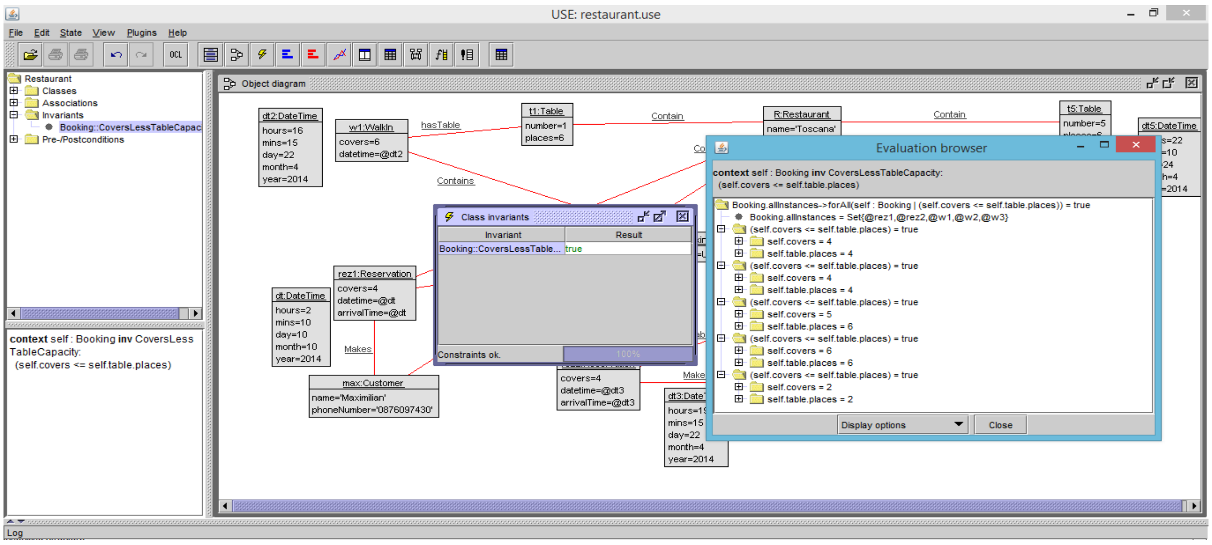
Constraints ok. 100%

# Invariants

1. Number of Covers Less than Table Capacity

```
context Booking
    inv CoversLessTableCapacity:
        self.covers <= self.table.places
```

This will be true if all tables have less covers than their capacity. For my object diagram from above, this invariant will be true.



context self : Booking inv CoversLessTableCapacity:
(self.covers <= self.table.places)

Class invariants

| Invariant | Result |
|---|---|
| Booking::CoversLessTable... | true |

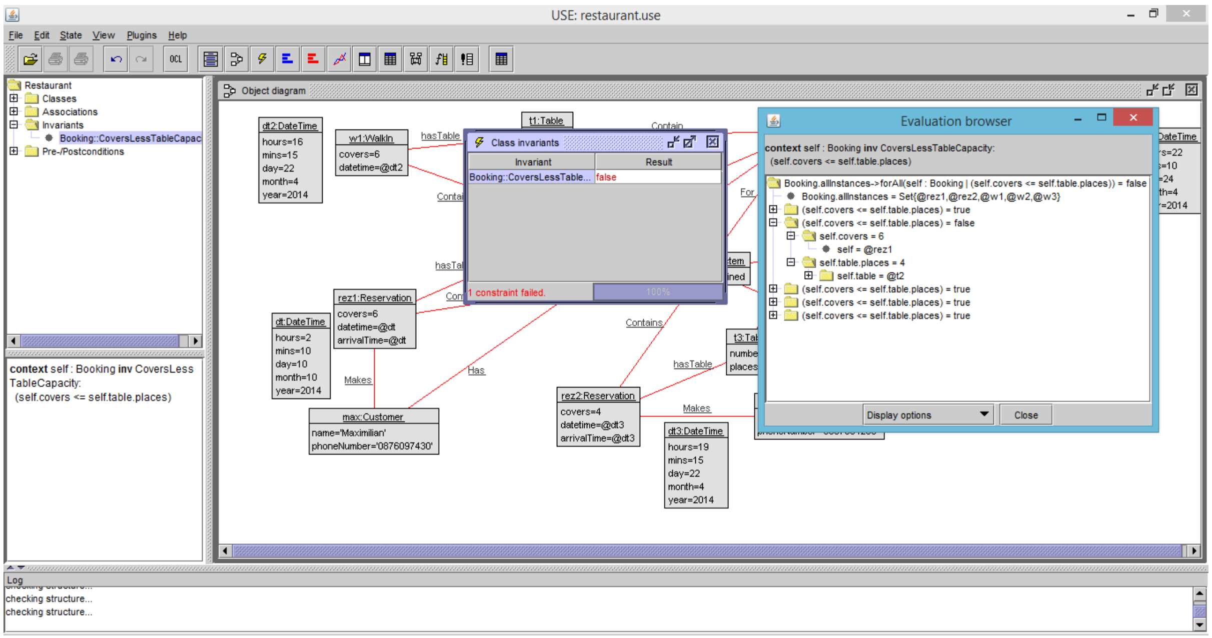
Constraints ok. 100%

Evaluation browser

```
context self : Booking inv CoversLessTableCapacity:
 (self.covers <= self.table.places)

Booking.allInstances->forAll(self : Booking | (self.covers <= self.table.places)) = true
   Booking.allInstances = Set{@rez1,@rez2,@w1,@w2,@w3}
   (self.covers <= self.table.places) = true
      self.covers = 4
      self.table.places = 4
   (self.covers <= self.table.places) = true
      self.covers = 4
      self.table.places = 4
   (self.covers <= self.table.places) = true
      self.covers = 5
      self.table.places = 6
   (self.covers <= self.table.places) = true
      self.covers = 6
      self.table.places = 6
   (self.covers <= self.table.places) = true
      self.covers = 2
      self.table.places = 2
```

Display options  Close

I will now change one reservation to vane more covers than table capacity to see the difference. Rez1 has 4 covers and they have table 2 with 4 places. I will now increase the number of covers from that reservation so the invariant will be false.



```
context self : Booking inv CoversLessTableCapacity:
 (self.covers <= self.table.places)
```

| Invariant | Result |
|---|---|
| Booking::CoversLessTable... | false |

1 constraint failed.

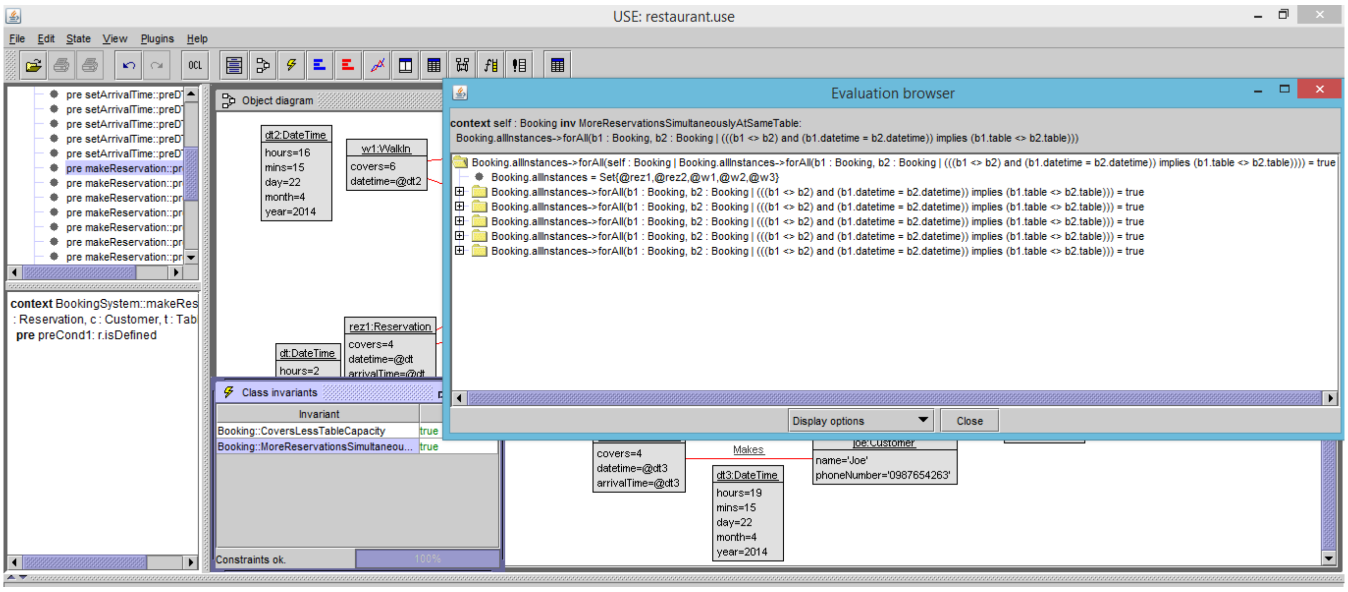
```
Booking.allInstances->forAll(self : Booking | (self.covers <= self.table.places)) = false
    Booking.allInstances = Set{@rez1,@rez2,@w1,@w2,@w3}
    (self.covers <= self.table.places) = true
    (self.covers <= self.table.places) = false
        self.covers = 6
            self = @rez1
        self.table.places = 4
            self.table = @t2
    (self.covers <= self.table.places) = true
    (self.covers <= self.table.places) = true
    (self.covers <= self.table.places) = true
```

```
checking structure...
checking structure...
```

2. Another invariant could be: There cannot be more reservations at the same time for the same table in the restaurant (It should be impossible to double book a table). This invariant looks like this:

```
context Booking
    inv CoversLessTableCapacity:
        self.covers <= self.table.places
    inv MoreReservationsSimultaneouslyAtSameTable:
        Booking.allInstances->forAll(b1,b2 | ((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))
```

This s showing the invariant value when it meets the requirements.



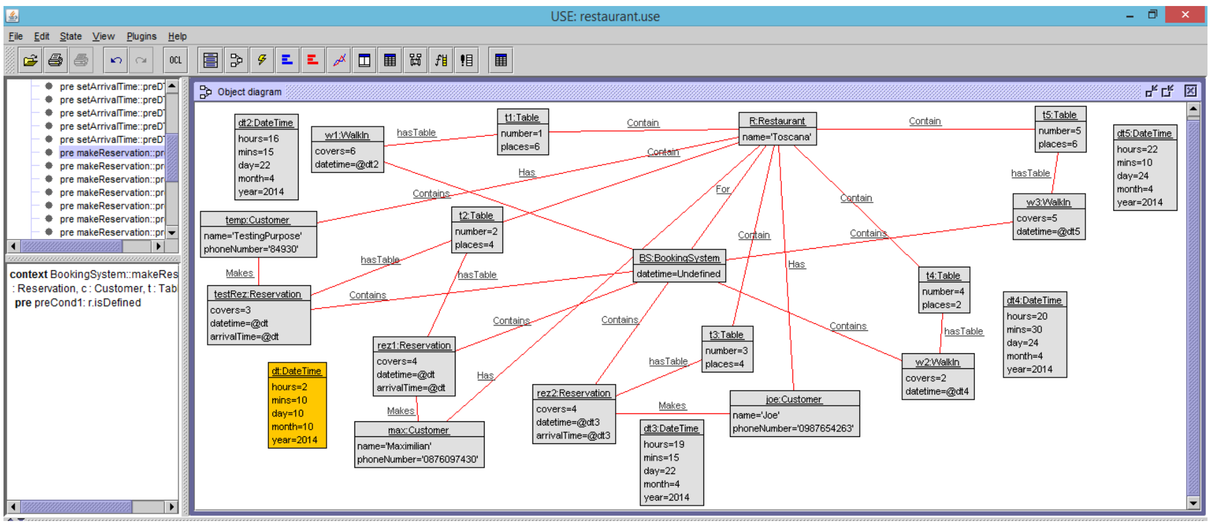
```
context self : Booking inv MoreReservationsSimultaneouslyAtSameTable:
 Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)))
```

```
Booking.allInstances->forAll(self : Booking | Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)))) = true
    Booking.allInstances = Set{@rez1,@rez2,@w1,@w2,@w3}
    Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))) = true
    Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))) = true
    Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))) = true
    Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))) = true
    Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table))) = true
```

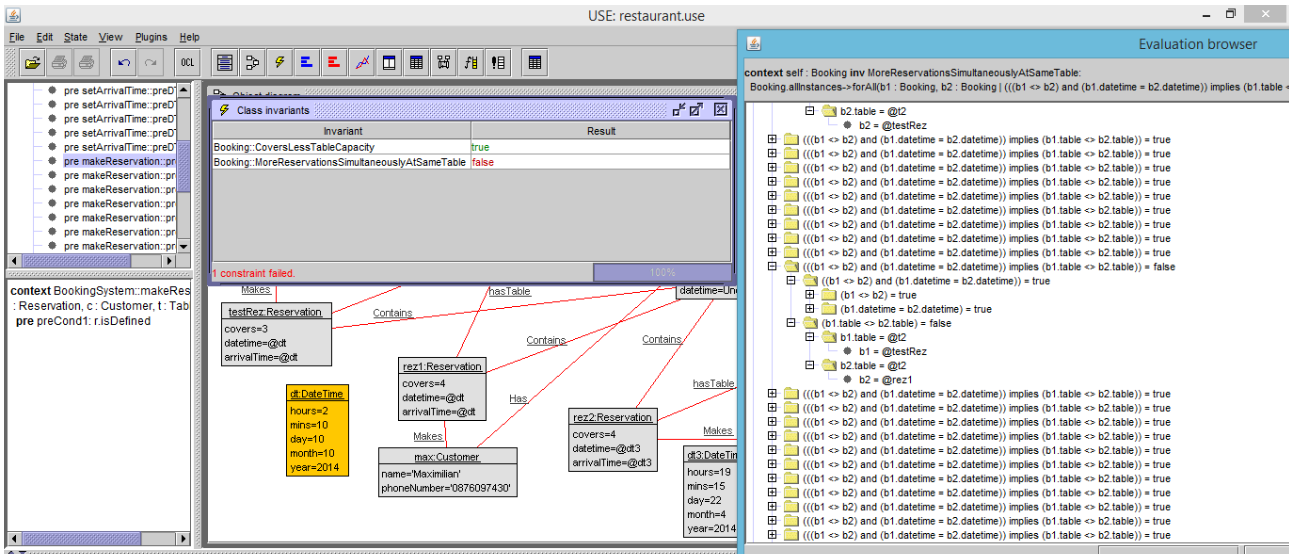
| Invariant | |
|---|---|
| Booking::CoversLessTableCapacity | true |
| Booking::MoreReservationsSimultaneou... | true |

Constraints ok.

To negate this invariant I will create a temporary customer that will make a reservation at the same time and table as max Customer. The invariant should be false this time.


USE: restaurant.use
File Edit State View Plugins Help
Object diagram
pre setArrivalTime::preDT
pre makeReservation::pr
context BookingSystem::makeRes : Reservation, c : Customer, t : Tab
pre preCond1: r.isDefined
dt2:DateTime hours=16 mins=15 day=22 month=4 year=2014
w1:WalkIn covers=6 datetime=@dt2
t1:Table number=1 places=6
R:Restaurant name='Toscana'
t5:Table number=5 places=6
dt5:DateTime hours=22 mins=10 day=24 month=4 year=2014
temp:Customer name='TestingPurpose' phoneNumber='84930'
t2:Table number=2 places=4
w3:WalkIn covers=5 datetime=@dt5
BS:BookingSystem datetime=Undefined
t4:Table number=4 places=2
testRez:Reservation covers=3 datetime=@dt arrivalTime=@dt
dt4:DateTime hours=20 mins=30 day=24 month=4 year=2014
rez1:Reservation covers=4 datetime=@dt arrivalTime=@dt
t3:Table number=3 places=4
dt:DateTime hours=2 mins=10 day=10 month=10 year=2014
w2:WalkIn covers=2 datetime=@dt4
rez2:Reservation covers=4 datetime=@dt3 arrivalTime=@dt3
joe:Customer name='Joe' phoneNumber='0987654263'
max:Customer name='Maximilian' phoneNumber='0876097430'
dt3:DateTime hours=19 mins=15 day=22 month=4 year=2014
hasTable
Contain
Contains
Has
For
Makes

I pointed on the object diagram what is wrong and what is making the invariant to be false.



USE: restaurant.use

Class invariants

| Invariant | Result |
|---|---|
| Booking::CoversLessTableCapacity | true |
| Booking::MoreReservationsSimultaneouslyAtSameTable | false |

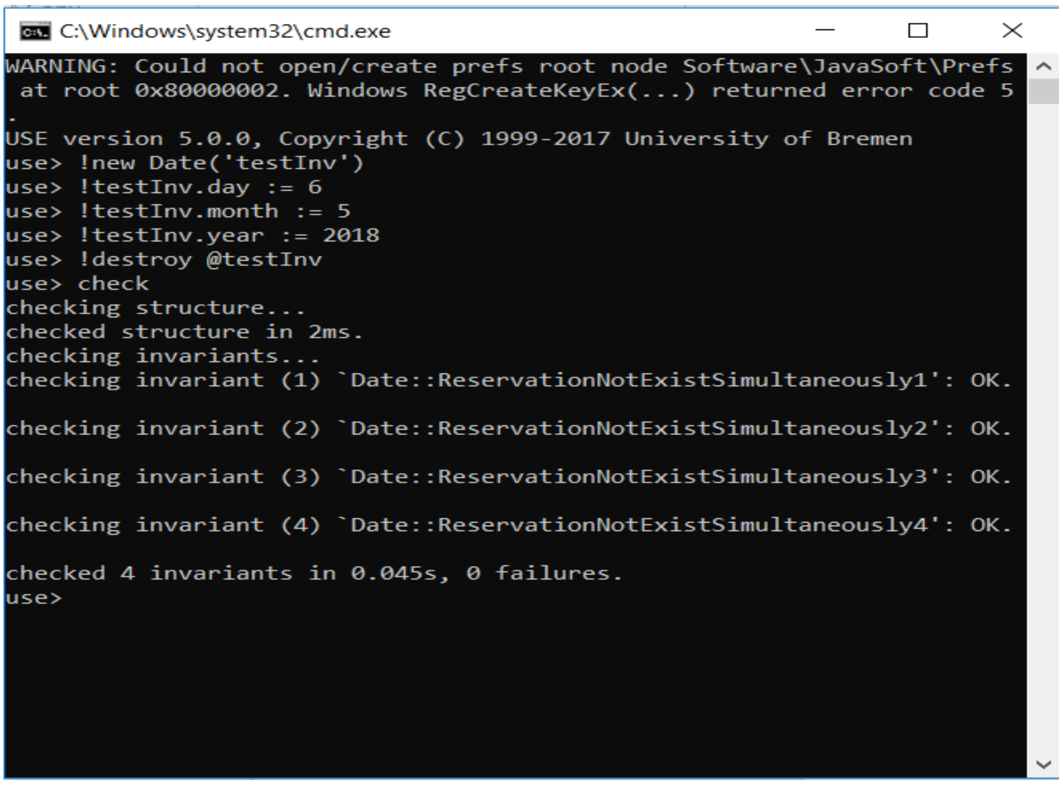
1 constraint failed. 100%

Evaluation browser

**context** self : Booking **inv** MoreReservationsSimultaneouslyAtSameTable:
Booking.allInstances->forAll(b1 : Booking, b2 : Booking | (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <

```
      b2.table = @t2
        b2 = @testRez
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = false
    ((b1 <> b2) and (b1.datetime = b2.datetime)) = true
      (b1 <> b2) = true
      (b1.datetime = b2.datetime) = true
    (b1.table <> b2.table) = false
      b1.table = @t2
        b1 = @testRez
      b2.table = @t2
        b2 = @rez1
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
  (((b1 <> b2) and (b1.datetime = b2.datetime)) implies (b1.table <> b2.table)) = true
```

All the Invariants can also be checked in command prompt:


C:\Windows\system32\cmd.exe

```
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs
 at root 0x80000002. Windows RegCreateKeyEx(...) returned error code 5
.
USE version 5.0.0, Copyright (C) 1999-2017 University of Bremen
use> !new Date('testInv')
use> !testInv.day := 6
use> !testInv.month := 5
use> !testInv.year := 2018
use> !destroy @testInv
use> check
checking structure...
checked structure in 2ms.
checking invariants...
checking invariant (1) `Date::ReservationNotExistSimultaneously1': OK.

checking invariant (2) `Date::ReservationNotExistSimultaneously2': OK.

checking invariant (3) `Date::ReservationNotExistSimultaneously3': OK.

checking invariant (4) `Date::ReservationNotExistSimultaneously4': OK.

checked 4 invariants in 0.045s, 0 failures.
use>
```

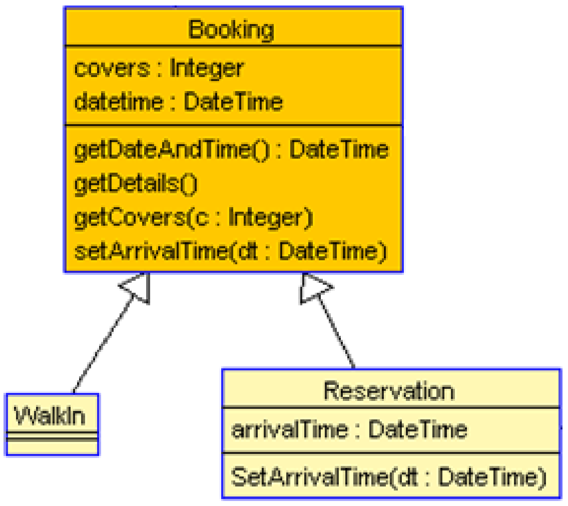
## Design Patterns

My restaurant model is assumed to be implemented as a desktop application: with single user and normal input and output devices like mouse, keyboard, screen etc.
In this model there is the Application Layer and Presentation layer. The Presentation layer it is used to present data to the application layer in an accurate, well-defined and standardized format, also is checking the application layer for updates. The presentation layer mainly translates data between the application layer and the network format. The Application layer is the one that directly interacts with the user.

### Adapter Design Pattern

Also known as Wrapper. Converts the interface of a class in another interface clients expect.
In my restaurant model I am using the Adapter pattern for the booking class. The Walk In and Reservation classes inherits from the Booking class so the system can be adapted. If no reservations are done for some tables when clients walk in the restaurant the walk in can be easily implemented instead of a reservation.
When a reservation is registered in the system, the user of this model has to create the reservation and the customer object in order to make the booking.

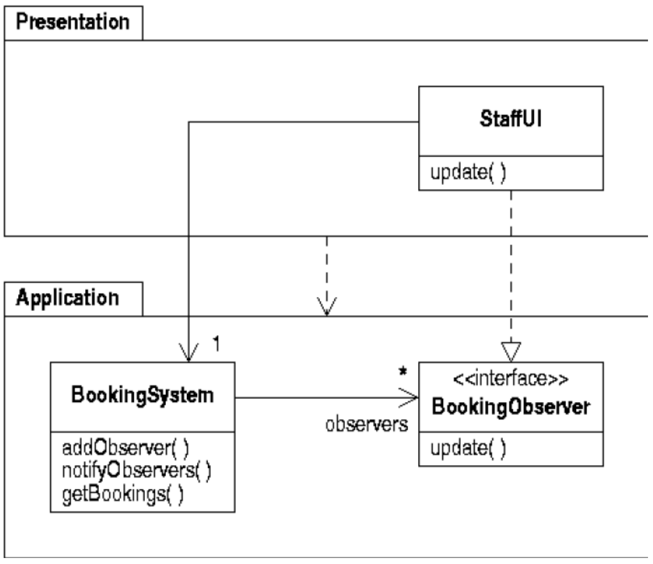
A reservation can be cancelled and instead of that, a walk in can be placed if exists



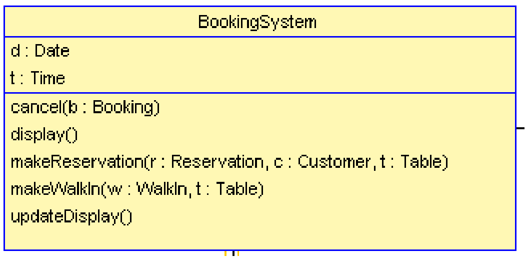
## Observer Design Pattern

The Observer pattern allows changes to one object, application, to be communicated to others by presentation without assuming what the other objects are.

The Observer Pattern Structure is:



In the Restaurant model, the Booking System class represents a whole-part hierarchy of objects.


BookingSystem
d : Date
t : Time
cancel(b : Booking)
display()
makeReservation(r : Reservation, c : Customer, t : Table)
makeWalkIn(w : WalkIn, t : Table)
updateDisplay()

Clients are able to ignore the difference between compositions of objects and individual objects. They will treat all objects in the composite structure uniformly.