

The Majestic Hotel,

A case for UML

Introduction

The Majestic hotel is in need for a system to help in the administration of their room rentals. The situation in the Majestic is currently as follows.

Rentable rooms

The hotel has a number of guest rooms for rent. They are divided over a number of floors. The more modern rooms have their own bathroom, but the older ones don't. For those rooms, there are bathrooms on each floor. The rooms are available in several classes: standard, business and executive. Executive rooms have all possible facilities. Of course, executive rooms have their own luxury bathroom. Standard rooms, on the other hand, are simple but much cheaper. The hotel also has a number of conference rooms, which can be hired by the day.

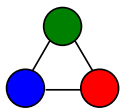
Booking and pricing

Each room is designated for a specific number of persons, but in some rooms it is possible to place an extra bed for a child. The pricing policy is one set price per room, plus an additional amount per guest, covering the cost for breakfast, etc.

Guests can either book a room in advance, or they may book a room on arrival if available.

Future plans

The hotel plans to upgrade their older rooms to a higher class in the coming years. This will raise the price of the rooms. The upgrade work will be planned in advance and take between two and six weeks per room.



Class diagram (domain version)

The class diagram for the Majestic Hotel is shown in figure 1 and 2. The complete model contains the following classes:

- **AdditionalItem**: something that can be reserved by a guest additional to a GuestRoom or ConferenceRoom, e.g. an extra bed, or an LCD projector.
- **Amount**: an amount of money to be paid.
- **BathRoom**: a room with bath, or shower and toilet.
- **ConferenceRoom**: a room for meetings and conferences.
- **Date**: a date, e.g. October 12, 1999.

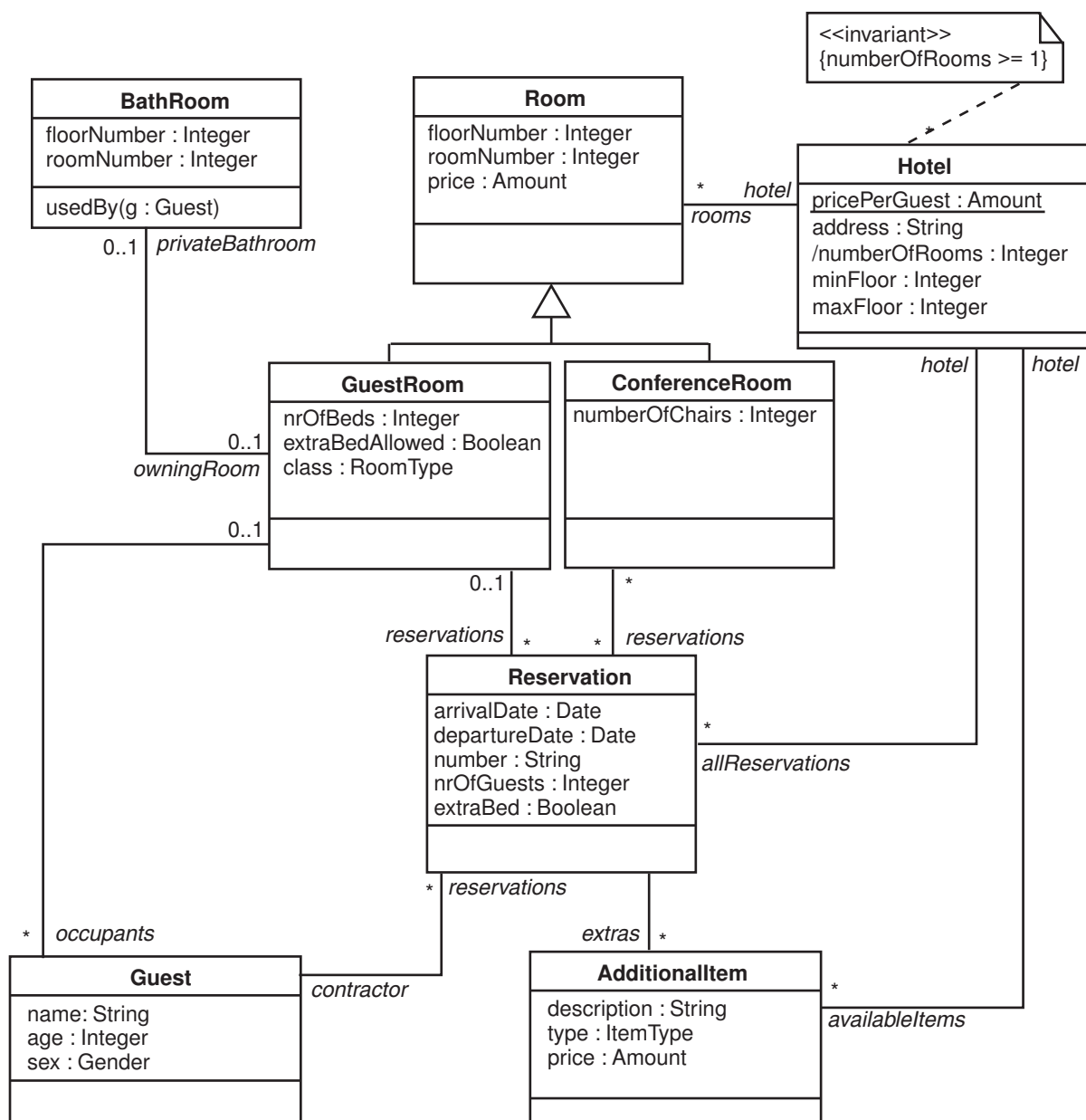
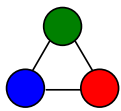


FIGURE 1. Domain class diagram



- Guest: a person who books a GuestRoom, or a company, institute or organisation that books a ConferenceRoom.
- GuestRoom: a bedroom with a number of beds.
- Hotel: a hotel that rents rooms to guests.
- Reservation: a reservation for a period of time of a room for a guest.

The utility classes are shown in figure 2.

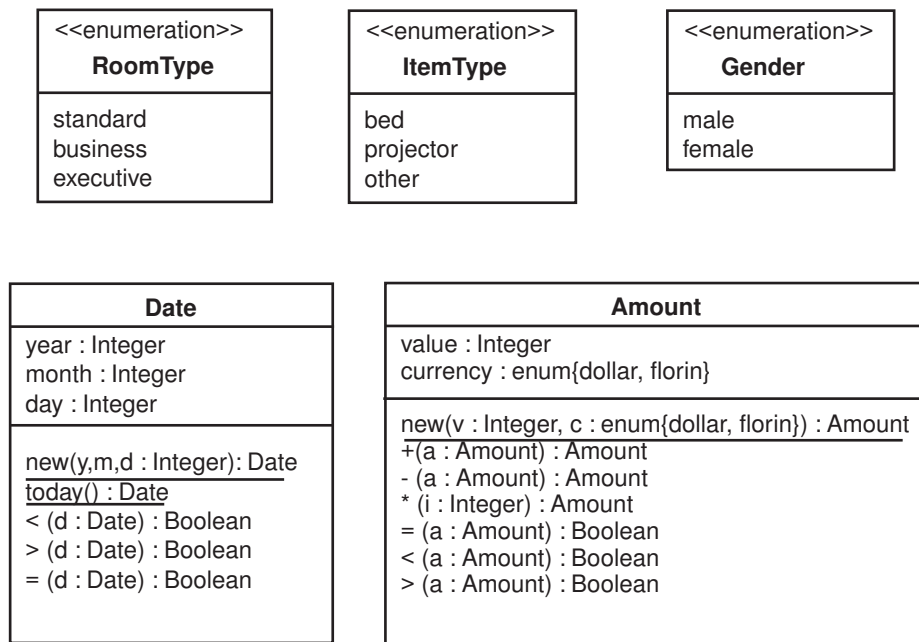
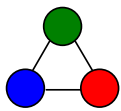


FIGURE 2. Utility classes



Basic business rules in OCL

1. A hotel has at least one room

```
context Hotel inv:  
  numberOfRooms >= 1
```

2. A hotel has at least three floors

```
context Hotel inv:  
  maxFloor - minFloor >= 2
```

3. An executive room has at most two beds.

```
context GuestRoom inv:  
  class = RoomType::executive implies (nrOfBeds <= 2 and extraBedAllowed = false)
```

4. The arrival date must be before the departure date.

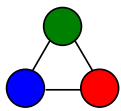
```
context Reservation inv:  
  arrivalDate < departureDate
```

5. The floor number of a room must be within the minimum and maximum floor numbers of the hotel.

```
context Room inv:  
  floorNumber <= hotel.maxFloor and floorNumber >= hotel.minFloor
```

6. Only adults may reserve a room

```
context Reservation inv:  
  self.contractor.age >= 18
```



Basic business rules in OCL with collections

1. A Reservation should have at least one room attached. Express this in OCL using the given class diagram. Could you improve the class diagram to make the OCL expression simpler?

```
context Reservation inv:
  GuestRoom->size() = 1 or ConferenceRoom->size() >= 1
```

2. The number of guests in a room must be less than or equal to the number of beds. If the room allows an extra bed, then an extra guest is allowed, but its age must be less or equal to four.

```
context GuestRoom inv:
  occupants->size() <= nrOfBeds or
  ( occupants->size() = nrOfBeds + 1 and extraBedAllowed and
    occupants->exists(o : Guest | o.age <= 4)
  )
```

3. The attribute *numberOfRooms* of Hotel is a derived attribute. Describe how the value can be derived as an invariant.

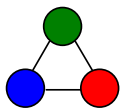
```
context Hotel
  inv: numberOfRooms = rooms->size()
```

4. The number of guest rooms in the hotel that allow an extra bed may not be more than 50% of the total number of guest rooms. Use an extra association from Hotel to GuestRoom called *guestrooms* with multiplicity 0..*.

```
context Hotel
  inv: guestrooms->select(extraBedAllowed)->size()/guestrooms->size() <= 0.50
```

5. The additional items that were booked with a reservation that starts today, should all be in the set of available items.

```
context Hotel
  inv: allReservations->select(arrivalDate = Date::today())->collect(extras)->
    forAll( i: AdditionalItem | self.availableItems->includes( i ) )
```



Basic business rules for class diagram

1. The attribute *numberOfRooms* of Hotel is a derived attribute. Describe how the value can be derived.

```
context Hotel::numberOfRooms
derive: rooms->size()
```

2. Define the attribute *currentGuests* to be the set of all current guests that stay at the hotel. Take as type of this attribute: Set(Guest).

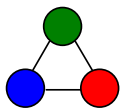
```
context Hotel
def: currentGuests : Set(Guest) = allReservations.guestRoom.occupants->asSet()
```

3. The initial value for price in Room is 50.

```
context Room::price
init: 50
```

4. The person that makes a reservation for a guest room should be one of the occupants. Clarify the cycle in the class diagram.

```
context Reservation
inv: GuestRoom->size() = 1 implies GuestRoom.occupants->includes(contractor)
```



Statechart diagrams

A reservation can be in a number of different states:

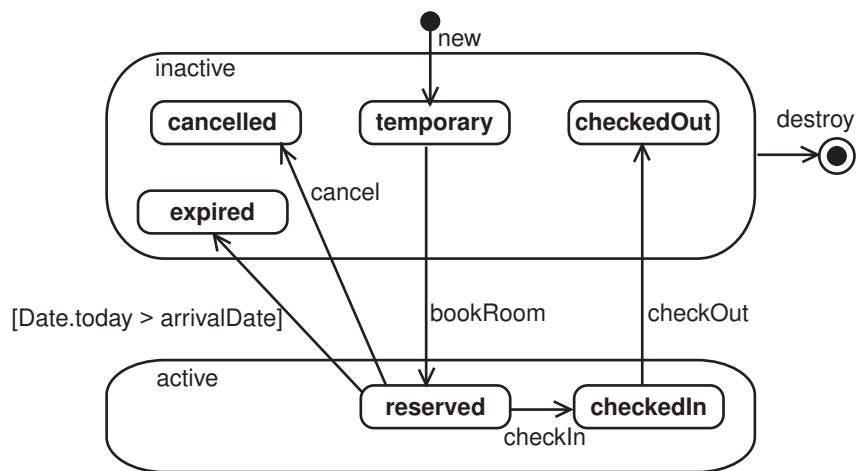
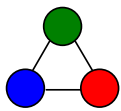


FIGURE 3. Statechart for Reservation



Advanced OCL

1. The operation *isAvailable(start : Date, end : Date) : Boolean* in class *GuestRoom* results in true if a room is available for the period between *start* and *end*.

Describe the result of this operation as a postcondition. Use the operation *overlapsWith(start, end : Date) : Boolean* from class *Reservation*.

```
context Room::isAvailable(start : Date, end : Date) : Boolean
post: result = self.reservations->forall( not overlapsWith(start, end))
```

2. The bathroom connected to a room may only be used by guests of that room. Describe the result of this operation as a postcondition to operation *usedBy(g: Guest)*.

```
context BathRoom::usedBy(g : Guest)
pre: self.owningRoom->notEmpty() implies owningRoom.occupants->includes(g)
```

3. The bathroom on a specific floor may only be used by guests of rooms on that floor. Again, describe the result of this operation as a postcondition to operation *usedBy(g: Guest)*.

```
context BathRoom::usedBy(g : Guest)
pre: self.owningRoom->isEmpty() and g.guestRoom->notEmpty() implies
      g.guestRoom.floorNumber = self.floorNumber
```

4. Upgrading a room will make it a higher class and the number of beds of the upgraded room is equal or less than the original room. Describe as postcondition.

```
context GuestRoom::upgrade()
post: nrOfBeds <= nrOfBeds@pre
post: class@pre = RoomType::business implies class = RoomType::executive
post: class@pre = RoomType::standard implies
      (class = RoomType::business) or (class = RoomType::executive)
```

5. A guest who does more than three reservations in the year 1999 will get a discount. Add an operation *hasDiscount() : Boolean* to *Reservation* and describe the result as a postcondition.

```
Reservation::hasDiscount() : Boolean
post: result = contractor.reservation->select(arrivalDate.year = 1999)
      ->asSet()->size() > 3
```

6. Write the pre and postconditions for the operation *Reservation::checkIn()*

```
context Reservation::checkIn()
pre: oclInState(reserved)
post: oclInState(checkedIn)
```

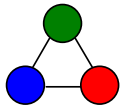
7. A separate bathroom that has been used ten times must be cleaned.

```
context BathRoom::usedBy(g : Guest)
post: timesUsed = timesUsed@pre + 1
post: timesUsed >= 10 implies mustClean

context Bathroom::clean()
post: (timesUsed = 0) and
      (mustClean = false)
```

8. Find all reservations that should be marked expired but are not marked. Define *expired : Set(Reservation)* as a derived attribute for *Hotel*.

```
context Hotel::expired : Set(Reservation)
derive: allReservations->select(
      oclInState(reserved) and arrivalDate > Date.today())
```



9. The number of guestrooms is greater than the number of conference rooms.

```
context Hotel inv:
```

```
rooms->select(oclIsKindOf(GuestRoom)) >
```

```
rooms->select(oclIsKindOf(ConferenceRoom))
```