The Little Inn

Andreas Dan Petersen, Daniel Blom & Christoffer Frederiksen - Dat18C

Content

- Design
  - Requirements
  - Use cases
  - Use case diagram
  - Domain model
  - Class Diagram - First Try
  - Class Diagram - Restaurant
  - Class Diagram - Accommodation & Staff
- Kode
  - Inn startup + mock data
  - Working hours + check in/out
  - Renting rooms + keeping records

Requirements

Software Responsibilities

- Working hours for staff
- Commodity storage
- Room reservation
- Meal orders
- Recipes

Requirements

Key Stakeholders	**Criteria for success**
Guest	● place meal order ● book room ● checkout
Staff	● check in/out ● lookup work schedule
Chef	● lookup meal orders ● register prepared meal
Servant	● take and serve orders ● take payment
Receptionist	● manage accommodation
Supervisor	● manage work schedules

Use Cases

Reserve room

No: 5
Stakeholder: Guest
Actor: Receptionist
2nd Actor: ReservationBook
Scenario:

The Guest asks a Receptionist for a room(s) for a period of days. The Receptionist places the reservation in the ReservationBook.

Precondition:
Success guarantee:

The amount of rooms needed must be available in the period of days wanted.
Extensions: **a)** Room(s) not available -> Pick another period of days.
Frequency: As often as a Guest needs a room(s)
Misc.:

Use Cases

Staff check in

No: 6
Actor: Staff
2nd Actor: WorkTracker
Scenario: Staff tells the WorkTracker that the staff member has met.
Precondition:
Success guarantee:
Extensions:
Frequency: Every day the Staff member meets.
Misc.:

Use Cases

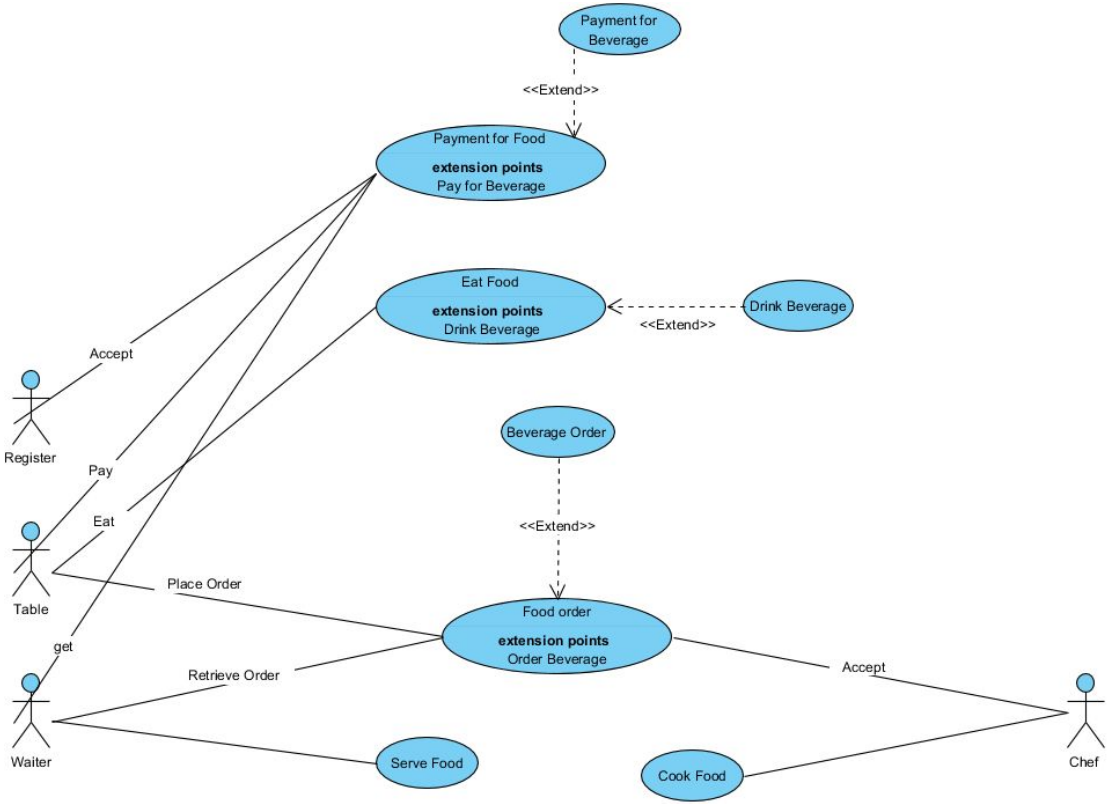
Staff check out

No: 7
Actor: Staff
2nd Actor: WorkTracker
Scenario: Staff tells the WorkTracker that the staff member is leaving.
Precondition: Staff check in (6)
Success guarantee:

Staff must have checked in to check out.
Extensions: **a)** Staff not checked in -> Tell staff that.
Frequency: As often as the staff member must meet.
Misc.:

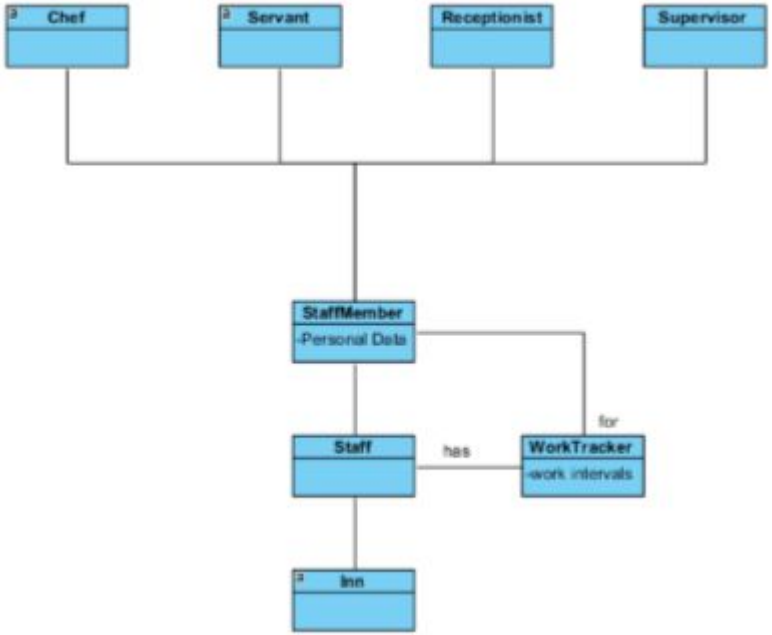
Use Case Diagram

Meal Order



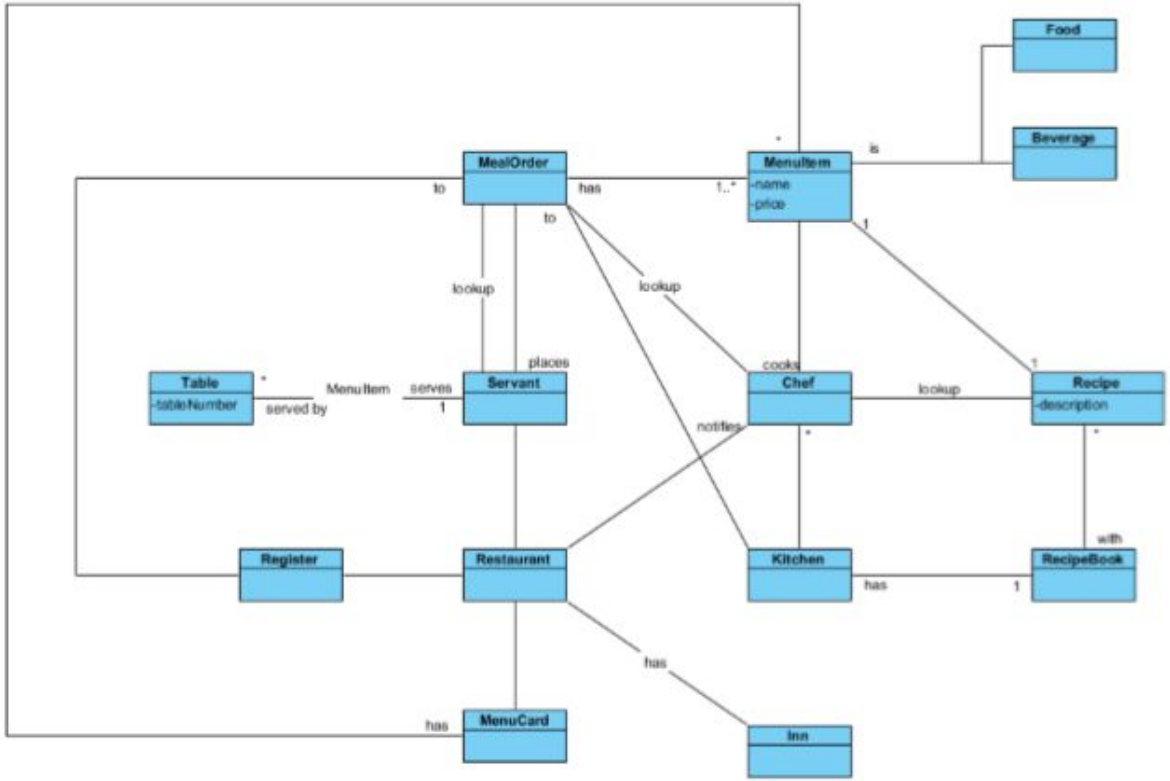
Domain Model

Staff



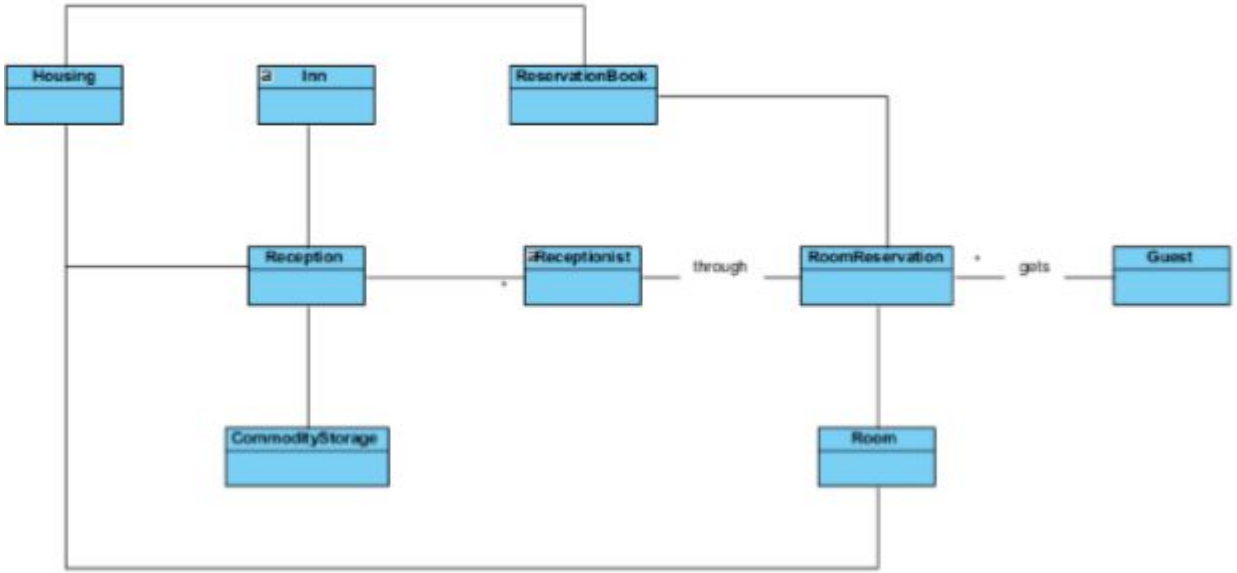
Domain Model

Restaurant



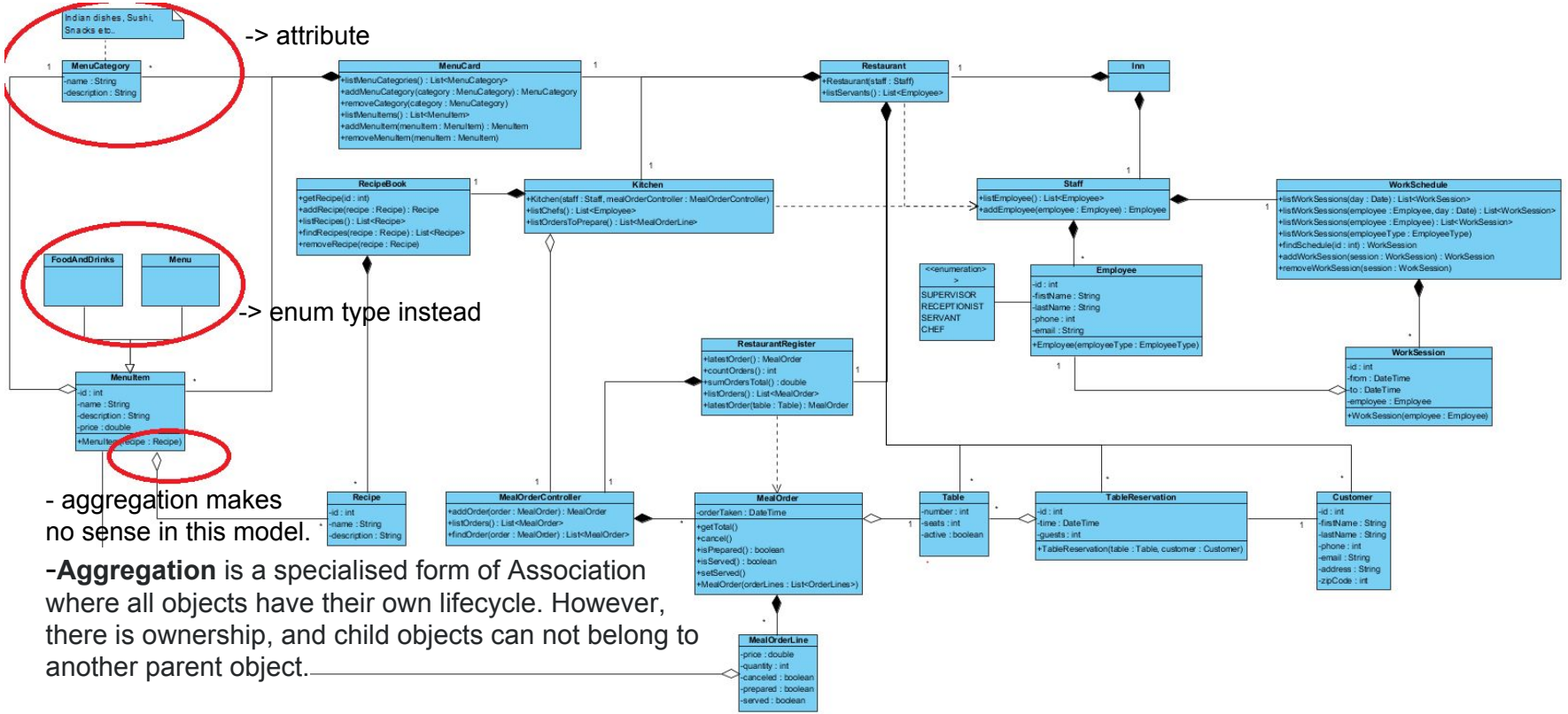
Domain Model

Reception



Class diagram

Inn - w.o. Reception

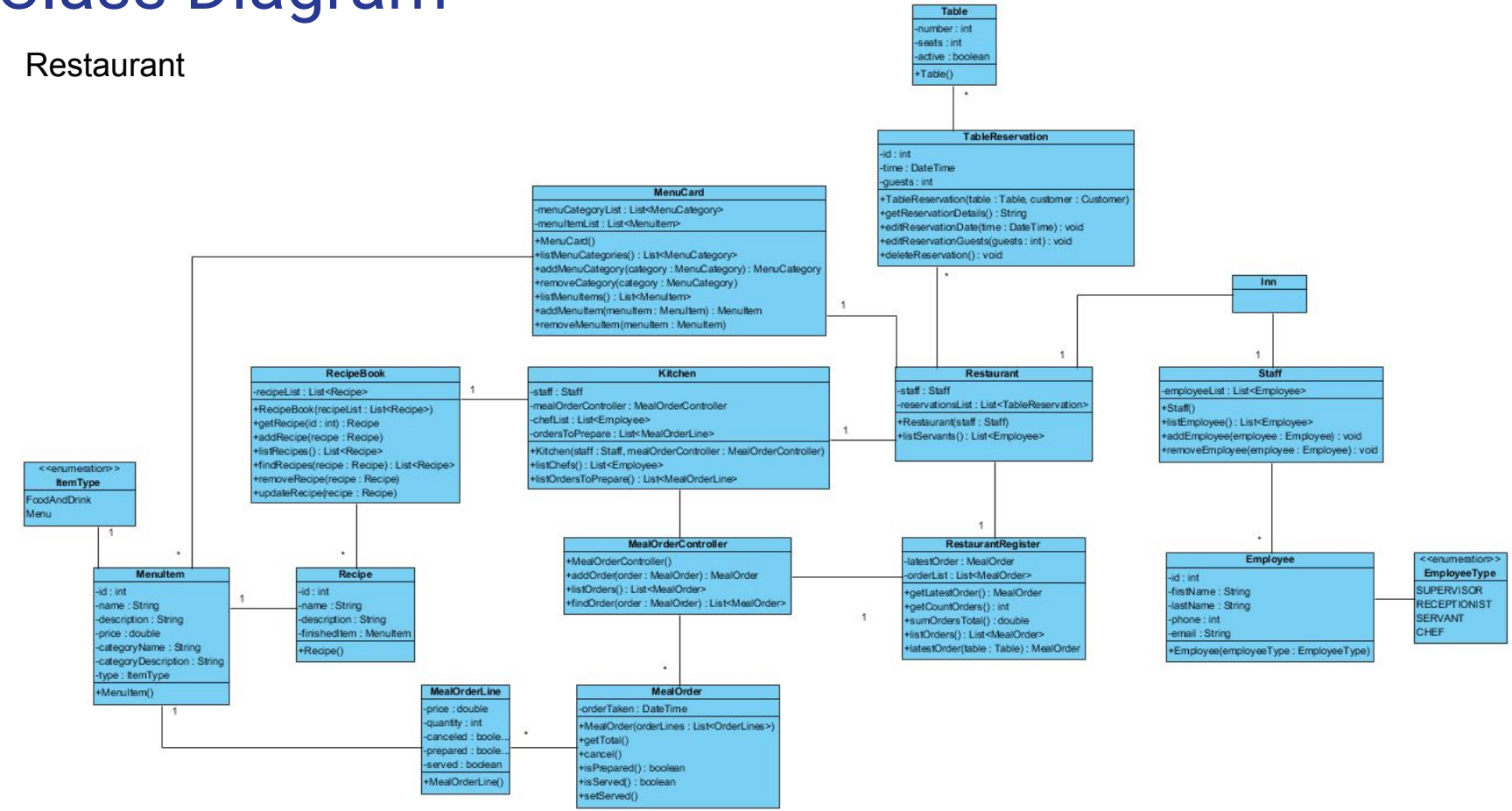


- aggregation makes no sense in this model.

-**Aggregation** is a specialised form of Association where all objects have their own lifecycle. However, there is ownership, and child objects can not belong to another parent object.

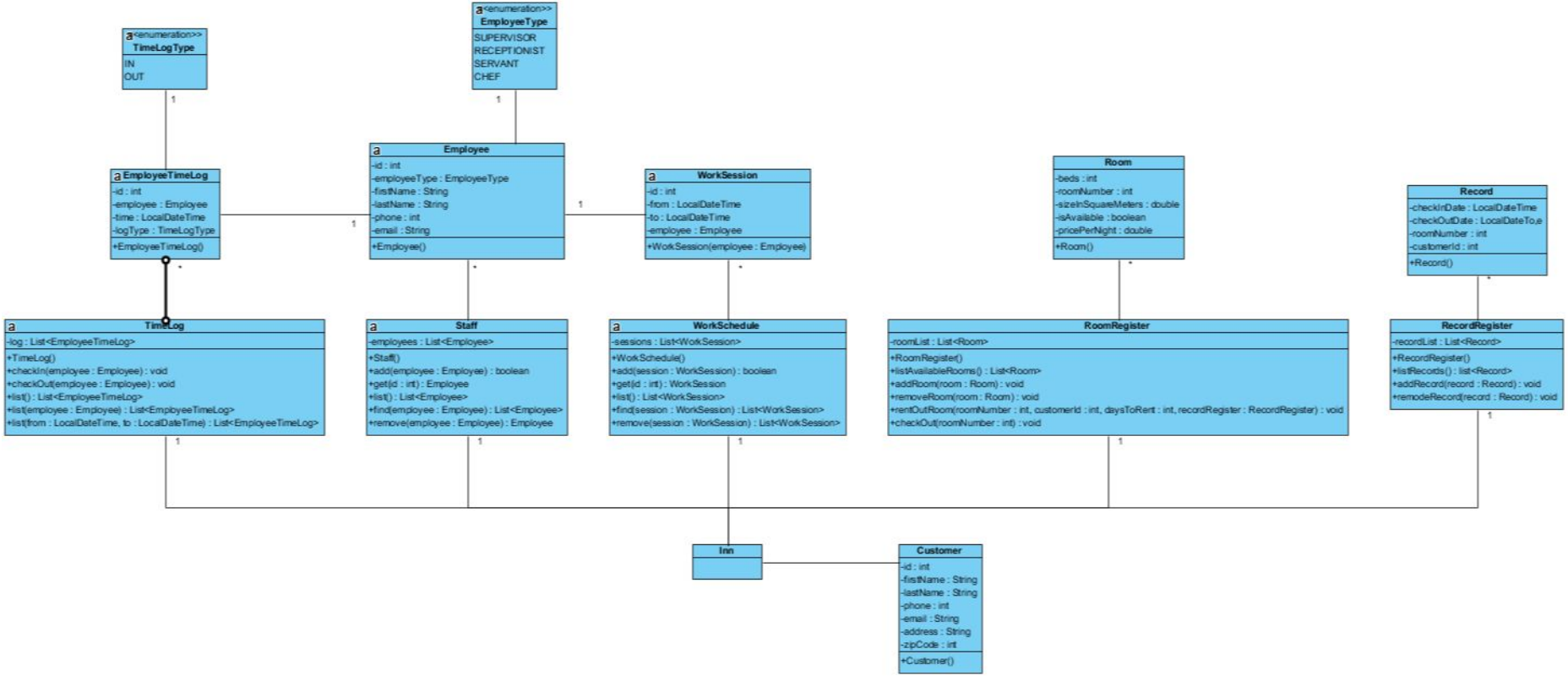
Class Diagram

Restaurant



Class Diagram

Accommodation & Staff



Code

Inn

- startup

```java
import java.util.Scanner;
import java.time.LocalDateTime;

public class Inn {
    public static void main(String[] args)
    {
        Staff staff = new Staff();                          // employees
        TimeLog timeLog = new TimeLog();                    // employee timestamping
        WorkSchedule workSchedule = new WorkSchedule();     // employee work schedule
        RoomRegister roomRegister = new RoomRegister();     // rooms
        RecordRegister recordRegister = new RecordRegister(); // room reservations

        //testing customers
        Customer[] customers = new Customer[3];
        customers[0] = new Customer(0, "Anders", "Andersen", 55555555, "mail@gmail.com", "adressevej 10", 2880);
        customers[1] = new Customer(1, "Jens", "Jensen", 44444444, "email@gmail.com", "vejnavn 19", 2690);
        customers[2] = new Customer(2, "Peter", "Petersen", 77777777, "email@hotmail.com", "addresse 2", 2540);

        //testing rooms
        roomRegister.addRoom(new Room(1, 2, 19.2, true, 289.0));
        roomRegister.addRoom(new Room(2, 3, 25.4, true, 359.0));
        roomRegister.addRoom(new Room(3, 1, 11.9, true, 189.0));

        //testing staff
        staff.add(new Employee(0, "John", "Brown", 63468412, "John1974@gmail.com", EmployeeType.SUPERVISOR));
        staff.add(new Employee(0, "Jane", "Brown", 63468412, "jane1976@gmail.com", EmployeeType.RECEPTIONIST));
        staff.add(new Employee(0, "Carl", "Carlson", 63468412, "carlc1988@gmail.com", EmployeeType.CHEF));
        staff.add(new Employee(0, "Lenny", "Leonard", 63468412, "lenny1988@gmail.com", EmployeeType.SERVANT));

        //scanner to read input
        Scanner scanner = new Scanner(System.in);

        //bool to check if program should continue running
        boolean runProgram = true;

        System.out.println("*** Inn system loaded ***");
```

Code

Inn

- Menu

```
*** Inn system loaded ***
Keys:
0 - Exit the program
1 - TimeLog
2 - Manage the reception
3 - Manage Staff w. working hours (to be implemented)
4 - Manage Customers (to be implemented)
Input>
```

```
while (runProgram)
{
    //read subsystem input choice
    System.out.print("Keys:\n0 - Exit the program"
        + "\n1 - TimeLog\n2 - Manage the reception"
        + "\n3 - Manage Staff w. working hours (to be implemented)"
        + "\n4 - Manage Customers (to be implemented)"
        + "\nInput>");
    int subSystemChoice = scanner.nextInt();

    if (subSystemChoice == 0)
    {
        //end program at next while check
        runProgram = false;
    }
```

Code

Inn

- Timelog
  check in/out

```
*** TimeLog ***
Keys:
0 - Exit the timelog sub system
1 - check in
2 - check out
3 - List records
Input>1
id: 1   firstName: John         lastName: Brown
id: 2   firstName: Jane         lastName: Brown
id: 3   firstName: Carl         lastName: Carlson
id: 4   firstName: Lenny        lastName: Leonard
Chose your ID
Input>3
Carl checked in.
*** TimeLog ***
Keys:
0 - Exit the timelog sub system
1 - check in
2 - check out
3 - List records
Input>
```

```
else if (subSystemChoice == 1) // time log part of the system
{
    boolean timeLogChosen = true;
    while (timeLogChosen) //makes the menu stay inside the reception loop
    {
        System.out.print("*** TimeLog ***\nKeys:\n0 - Exit the timelog sub system"
            + "\n1 - check in"
            + "\n2 - check out"
            + "\n3 - List records"
            + "\nInput>");
        int timeLogChoice = scanner.nextInt();
        if(timeLogChoice == 0) {
            timeLogChosen = false;
        } else if(timeLogChoice == 1) {
            for(Employee emp : staff.list()) {
                System.out.println(emp.toString());
            }

            System.out.print("Chose your ID\nInput>");
            int empIdChosen = scanner.nextInt(); // get input
            Employee emp = staff.get(empIdChosen);

            if(emp != null) {
                timeLog.checkIn(emp);
                System.out.println(emp.getFirstName() +" checked in.");
            }
        } else if(timeLogChoice == 2) {
```

Code

Inn

- Timelog list

```
*** TimeLog ***
Keys:
0 - Exit the timelog sub system
1 - check in
2 - check out
3 - List records
Input>3
id: 1   firstName: John         lastName: Brown         ph
id: 2   firstName: Jane         lastName: Brown         ph
id: 3   firstName: Carl         lastName: Carlson       ph
id: 4   firstName: Lenny        lastName: Leonard       ph
Chose your ID
Input>3
TimeLog for: Carl
id: 1   time: 2018-10-25T14:11:09.856392900     type: In
id: 2   time: 2018-10-25T14:11:53.770067200     type: Out

*** TimeLog ***
Keys:
0 - Exit the timelog sub system
1 - check in
2 - check out
3 - List records
Input>
```

```
} else if(timeLogChoice == 3) {
   for(Employee emp : staff.list()) {
      System.out.println(emp.toString());
   }

   System.out.print("Chose your ID\nInput>");
   int empIdChosen = scanner.nextInt(); // get input
   Employee emp = staff.get(empIdChosen);

   if(emp != null) {
      System.out.println("TimeLog for: "+ emp.getFirstName());
      for(EmployeeTimeLog tLog : timeLog.list(emp)) {
         String inOrOut = tLog.getLogType() == TimeLogType.IN ? "In" : "Out";
         System.out.println("id: "+ tLog.getId()
         + "\ttime: "+ tLog.getTime().toString()
         + "\ttype: "+ inOrOut);
      }
      System.out.println();
   }
}
```

Code

Inn

- Reception
  List rooms

```
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>1
Room nr.: 1 Beds: 2 Size: 19.2m2 Is available: true Price per night: 289.0
Room nr.: 2 Beds: 3 Size: 25.4m2 Is available: true Price per night: 359.0
Room nr.: 3 Beds: 1 Size: 11.9m2 Is available: true Price per night: 189.0
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
```

```
else if (subSystemChoice == 2) //reception menu
{
   boolean receptionChosen = true;

   while (receptionChosen) //makes the menu stay inside the reception loop
   {
      System.out.print("*** Reception ***\nKeys:\n0 - Exit the reception sub system"
         + "\n1 - List all rooms"
         + "\n2 - List all available rooms"
         + "\n3 - List all records"
         + "\n4 - Rent out a room"
         + "\n5 - Check-out of room"
         + "\nInput>");
      int receptionChoice = scanner.nextInt();

      if (receptionChoice == 0) //exit the reception loop and enter main loop
      {
         receptionChosen = false;
      }
      else if (receptionChoice == 1) //list all rooms
      {
         if (roomRegister.listRooms().size() > 0)
         {
            for (Room room : roomRegister.listRooms())
            {
               System.out.println(room.toString());
            }
         }
         else
         {
            System.out.println("No rooms to list.");
         }
      }
```

Code

Inn

- Reception book
  room + checkout

```
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>4
Room nr.>2
Customer ID>1
How many days will the room be rented out?>3
Room succesfully rented out. Record information:
Check-in date: 2018-10-25T14:33:38.671800200 Check-out date: 2
018-10-28T11:00 Room nr.: 2 Customer ID: 1
Price to pay for the stay is 1077.0
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>
```

```
else if (receptionChoice == 4) //rent out a room
{
   System.out.print("Room nr.>");
   int roomNumber = scanner.nextInt();
   System.out.print("Customer ID>");
   int customerId = scanner.nextInt();
   System.out.print("How many days will the room be rented out?>");
   int daysToRent = scanner.nextInt();

   roomRegister.rentOutRoom(roomNumber, customerId, daysToRent, recordRegister);
}
else if (receptionChoice == 5) //checkout customer from room number
{
   System.out.print("Room nr.>");
   int roomNumber = scanner.nextInt();
   roomRegister.checkOut(roomNumber);
}
else
{
   System.out.println("Please select either 0, 1, 2, 3, 4 or 5 to continue...");
}
```

Code

Inn

- Reception list available rooms

```
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>2
Room nr.: 1 Beds: 2 Size: 19.2m2 Is available: true Price per night: 289.0
Room nr.: 3 Beds: 1 Size: 11.9m2 Is available: true Price per night: 189.0
```

```
else if (receptionChoice == 2) //list only rooms that are available
{
    if (roomRegister.listAvailableRooms().size() > 0)
    {
        for (Room room : roomRegister.listAvailableRooms())
        {
            System.out.println(room.toString());
        }
    }
    else
    {
        System.out.println("No available rooms to list.");
    }
}
```

Code

Inn

- Reception checkout
  + available rooms

```
5 - Check-out of room
Input>5
Room nr.>2
Room nr.: 2 checked out and made available.
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>2
Room nr.: 1 Beds: 2 Size: 19.2m2 Is available: true Price per night: 289.0
Room nr.: 2 Beds: 3 Size: 25.4m2 Is available: true Price per night: 359.0
Room nr.: 3 Beds: 1 Size: 11.9m2 Is available: true Price per night: 189.0
```

```
else if (receptionChoice == 5) //checkout customer from room number
{
    System.out.print("Room nr.>");
    int roomNumber = scanner.nextInt();
    roomRegister.checkOut(roomNumber);
}
```

Code

Inn

- Reception checkout
  + available rooms

```
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>3
Check-in date: 2018-10-25T14:43:48.350801200 Check-out date: 2018-10-28T11:00 Room nr.: 2 Customer ID: 1
*** Reception ***
Keys:
0 - Exit the reception sub system
1 - List all rooms
2 - List all available rooms
3 - List all records
4 - Rent out a room
5 - Check-out of room
Input>
```

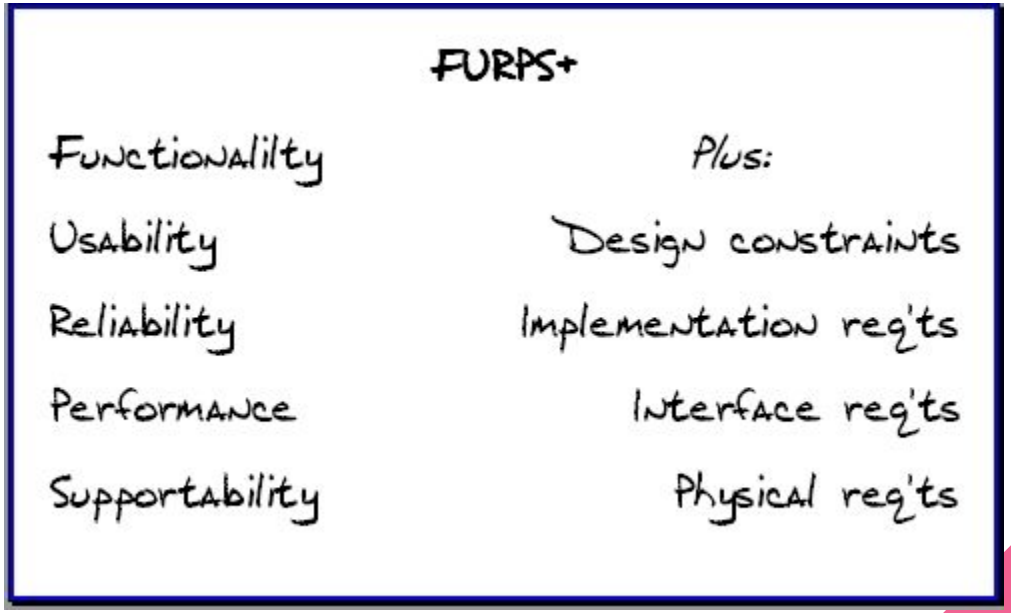
```
else if (receptionChoice == 3) //list all records
{
    if (recordRegister.listRecords().size() > 0)
    {
        for (Record record : recordRegister.listRecords())
        {
            System.out.println(record.toString());
        }
    }
    else
    {
        System.out.println("No records to list.");
    }
}
```

Reflection..

- Requirements
  - more FURPS+.
- Domain model
  - We didn’t correct the domain model.
- Class diagram
  - We learned alot about keeping it simple.
- Code
  - Instead of having the attribute ‘Available’ in ‘Room’, we could ask the ‘RecordRegister’ if the room with a given ‘Id’ is currently available or available in a given period. This helps keeping responsibility separated.
  - We could make UI-Controller classes to make the menu code more manageable.

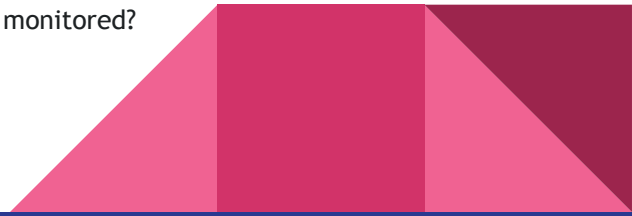
FURPS+

The FURPS+ acronym, devised by Robert Grady of HP, provides a bit more meat around what we mean by non-functional stories, and also provides a good way to categorize such needs. The breakdown here suggests some representative questions around potential needs.



FURPS+

- Functionality
  What the customer wants! Note that this includes security-related needs.
- Usability
  How effective is the product from the standpoint of the person who must use it? Is it aesthetically acceptable? Is the documentation accurate and complete?
- Reliability
  What is the maximum acceptable system downtime? Are failures predictable? Can we demonstrate the accuracy of results? How is the system recovered?
- Performance
  How fast must it be? What's the maximum response time? What's the throughput? What's the memory consumption?
- Supportability
  Is it testable, extensible, serviceable, installable, and configurable? Can it be monitored?



FURPS+

The + reminds us of a few additional needs that a customer could have:

- Design constraints
  Do things like I/O devices or DBMS constrain how the software must be built?
- Implementation requirements
  Do the programmers need to adhere to standards? Is the use of TDD required? Is statistically sound testing in the context of Cleanroom required?
- Interface requirements
  What downstream feeds must be created? What other systems must this one interface with? How frequent are feeds produced?
- Physical requirements
  What hardware must the system be deployable on? Must we be able to deploy to a machine no larger than 12" square, to be stationed in the Iraqi desert?