**Domain Model**

Domain model of current project might not look too complex, but in the same time it was challenging. As it is shown in the diagram, there are 10 classes: Reservation, Table, Order, OrderLine, Merchandise, Course, Snack, Drink, Staff and Waiter. To understand what each class is for, description and explanation of the most interesting fields are provided below:

**Reservation:** this class was made to allow the system to create and manage reservations for customers if they want to book a table in restaurant. Reservation can be created, found, updated and cancelled. It contains following fields – customersName, phoneNo, reservationDate, numberOfGuests, reservationRegistrationDate, reservedTime. When reservation is made depending on time and date table(-s) will change its (their) state in field isAvailable to false (indicating that it can’t be reserved for that specific time, latest time it can be booked will be one hour before reservation takes place and only if there are no more free tables available).

**Table:** this class was made to allow the system to create and manage the tables in restaurant. Table can be created, updated, found, and deleted. It contains the following fields – noOfSeats, tableNo, isAvailable, exists, tableOnTheNorth, tableOnTheEast, tableOnTheSouth, tableOnTheWest.

* isAvailable – provides the field to store the state of the table, either it is available or taken.
* exists – provides the field to store the state of the table, either it exists or not, if it’s no more available in the restaurant (removed), so it would not mess up the system in the history of orders and reservations. It will stay in the system until company decides to clean it up.
* tableOnTheNorth – provides the field to store information of the table to the north of current table so the system would know how to combine table reservations if there are more customers than the seats at the table and could find if there is available table next to it which could be also reserved
* tableOnTheEast – same as tableOnTheNorth only checks for available table to the east
* tableOnTheSouth - same as tableOnTheNorth only checks for available table to the south
* tableOnTheWest - same as tableOnTheNorth only checks for available table to the west

**Order:** this class was made to allow the system to create orders when customers come to the restaurant. Order can be created, found, updated and cancelled. It contains the following fields – totalPrice, isPaid, isActive.

* isPaid – provides the field to store the state of the payment, either customer has paid or not.
* isActive – provides the field to allow the system to store the state of the order. When reservation is made, customer is able to make a preorder so the chef can start cook when customer arrives. isActive field will indicate the system about if the order should be visible to chef.

**OrderLine:** this class was made to allow the system to add the same item more than one time to the Order. Object of the OrderLine can’t exist without the Order. It contains the following field – quantity.

**Merchandise:** this class was made to allow the system to create, update, find and remove objects of the merchandise. It is an abstract class which means that whenever you create any of its subclasses it will provide them with the same fields that are implemented in the class Merchandise. It contains the following fields – name, price, exists.

* exists – provides the field to store the state of the merchandise, either it exists or not, if it’s no more available in the restaurant (removed), so it would not mess up the system in the history of orders. It will stay in the system until company decides to clean it up.

**Course:** this class was made to allow the system to create, update, find and remove objects of the Course. Course is one of the main product of the restaurant which allows to create and record dishes that restaurant provides to its customers. It is a subclass of the abstract class Merchandise. It contains the following fields – ingredients, isVegetarian.

* Ingredients – provides the field to store the String of ingredients that describes what the dish is made of. It was a good idea for this field in order if customer has any allergy and he would like to find out what is the dish made of. Waiter will be able to find it out with ease.

**Miscellaneous:** this class was made to allow the system to create, update, find and remove objects of the Miscellaneous. This object is created for any other products that bar / restaurant is selling besides the courses and drinks. It contains the following fields – quantityInStock, minQuantityInStock.

* minQuantityInStock – provides the field to store the number of minimum amount of products in stock before the system notifies staff that it should be reordered.

**Drink:** this class was made to allow the system to create, update, find and remove objects of the Drink. With this class system can store all the drinks that bar / restaurant is selling. It contains the following fields – quantityInStock, alcoholConcentration, minQuantityInStock.

* alcoholConcentration – provides the field to store the number of the alcohol concentration in the drink. It waiters with information in case the customer will ask. It also allows the system to filter nonalcoholic drinks or alcoholic drinks.
* minQuantityInStock – works the same way as in the class Miscellaneous.

**Staff:** this class was made to allow the system to create, update, find and remove objects of the Staff. It is a superclass for class Waiter. It contains the following fields – name, bankAccount, address, profession, phoneNo, cprNo, exists.

* profession – provides the field to store the taken position of staff member in the system
* exists – same as all the other classes with this field it indicates if the staff member still works in the bar / restaurant and is stored in history.

**Waiter:** this class was made to allow the system to create, update, find and remove objects of the Waiter. This class allows to add waiter to Order to provide it with information of who served it. It contains the following field – workTables.

* workTables – provides the field to store a list of tables that waiter is working on at his working hours.

**Associations**

Reservation class is associated with classes Table and Order to store the core information about every detail for that specific reservation. Reservation can be made by phone and also we use the same system if the customers comes to the restaurant and wants to receive services. In any way it will be a reservation of a table. Reservation can have 1 to many tables if there is more persons than seats at the table and they are next to each other multiple tables can be reserved for the same reservation. Table can have 0 to many reservations in the same day if no other choices are left.

Order class is associated with classes Waiter and OrderLine to store the information about the order and waiter who served it, if something goes wrong. Order and OrderLine is an aggregate, where Order is the root. The orderLine can’t exist without the order. Order can have 1 to many orderLines. OrderLine can have one and only one order. Order must contain one Waiter. Waiter can have 0 to many orders.

OrderLine class is associated with Merchandise class. OrderLine can have only one kind of merchandise with quantity one or more. Merchandise can have 0 to many orderLines.

Merchandise is generalized in three subclasses – Course, Drink and Miscellaneous.

Class Staff is associated with the class Waiter. Class Waiter is a subclass of the Staff class which makes the Staff a superclass.

**Conclusion**

Analysis of the business workflow and its requirements for the system gave this project a good overview of the structure and functionality. This led to domain model as it is shown and described in this section. This is the core plan for the classes and their fields for the code and gave all the necessary information for system to develop further. This shows the improved business workflow, which includes the system.

Description also helped to keep all members informed about the design of our system and to understand what the fields are for. This lets the members to be up to date if something changes or if anyone has some doubts about the domain model and how everything works. Also it lets to avoid discussions about the same things between members if they missed anything.