Restaurant order and account system

Assignment #1

U. PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Master in Informatics and Computing Engineering

Distribution and Integration Technologies

Mariana Duarte Guimarães, up201307777@fe.up.pt

Tiago Duarte Carvalho, up201504461@fe.up.pt

Faculty of Engineering of the University of Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

April 26, 2019

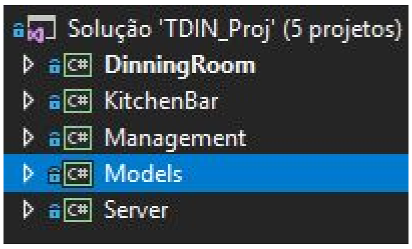
1. Introduction

The proposed assignment consists of developing an intranet distributed application using .NET remoting to automate a restaurant’s needs.
A restaurant needs to automatize the dining room orders, allowing them to be quickly communicated to the kitchen and bar tenders, to be prepared as soon as possible. Also, the same system should maintain a complete record of all orders, compute the bills of each table, and maintain a record of the total amount received in the day.
The architecture and implementation of our solution to this project will be described in the following sections.

2. Architecture

2.1 Description

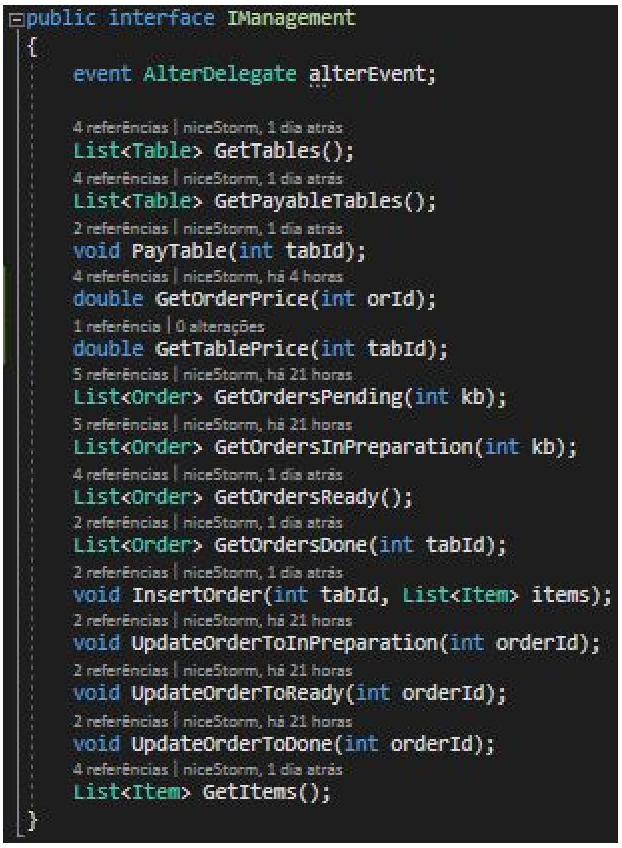
The implemented solution resides in a Client-Server architecture in which two clients communicate with the server.
Clients access the information in the server through methods, but also receive notifications of alterations in the server over events.
We have two clients, *DinningRoom*, and *KitchenBar*, that connect through remoting to the *Server* and are both WinForms, but also, two libraries *Models*, and *Management*.



```
Solução 'TDIN_Proj' (5 projetos)
  DinningRoom
  KitchenBar
  Management
  Models
  Server
```

2.2 Remote objects

Management is the remote object we use to implement the methods that handle the orders, tables and items and also trigger the events that update the information.



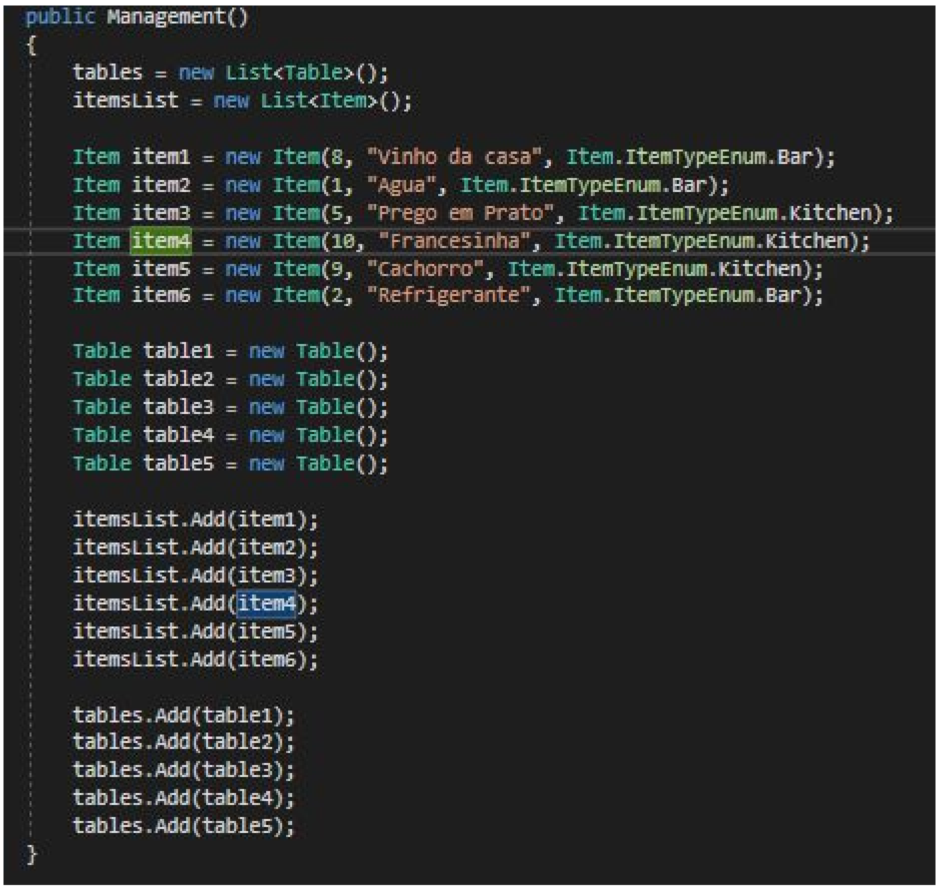
```
public interface IManagement
{
    event AlterDelegate alterEvent;

    4 referências | niceStorm, 1 dia atrás
    List<Table> GetTables();
    4 referências | niceStorm, 1 dia atrás
    List<Table> GetPayableTables();
    2 referências | niceStorm, 1 dia atrás
    void PayTable(int tabId);
    4 referências | niceStorm, há 4 horas
    double GetOrderPrice(int orId);
    1 referência | 0 alterações
    double GetTablePrice(int tabId);
    5 referências | niceStorm, há 21 horas
    List<Order> GetOrdersPending(int kb);
    5 referências | niceStorm, há 21 horas
    List<Order> GetOrdersInPreparation(int kb);
    4 referências | niceStorm, 1 dia atrás
    List<Order> GetOrdersReady();
    2 referências | niceStorm, 1 dia atrás
    List<Order> GetOrdersDone(int tabId);
    2 referências | niceStorm, 1 dia atrás
    void InsertOrder(int tabId, List<Item> items);
    2 referências | niceStorm, há 21 horas
    void UpdateOrderToInPreparation(int orderId);
    2 referências | niceStorm, há 21 horas
    void UpdateOrderToReady(int orderId);
    2 referências | niceStorm, há 21 horas
    void UpdateOrderToDone(int orderId);
    4 referências | niceStorm, 1 dia atrás
    List<Item> GetItems();
}
```

2.3 Methods

The remote object is responsible for the methods used by the client to access the objects described in the Models.

The methods are:

- Constructor of Management initializes some objects



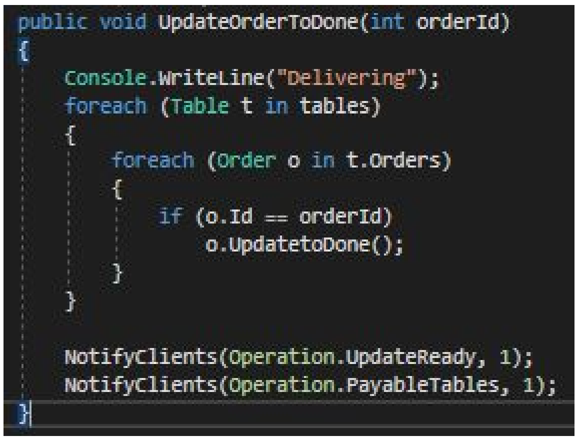
```
public Management()
{
    tables = new List<Table>();
    itemsList = new List<Item>();

    Item item1 = new Item(8, "Vinho da casa", Item.ItemTypeEnum.Bar);
    Item item2 = new Item(1, "Agua", Item.ItemTypeEnum.Bar);
    Item item3 = new Item(5, "Prego em Prato", Item.ItemTypeEnum.Kitchen);
    Item item4 = new Item(10, "Francesinha", Item.ItemTypeEnum.Kitchen);
    Item item5 = new Item(9, "Cachorro", Item.ItemTypeEnum.Kitchen);
    Item item6 = new Item(2, "Refrigerante", Item.ItemTypeEnum.Bar);

    Table table1 = new Table();
    Table table2 = new Table();
    Table table3 = new Table();
    Table table4 = new Table();
    Table table5 = new Table();

    itemsList.Add(item1);
    itemsList.Add(item2);
    itemsList.Add(item3);
    itemsList.Add(item4);
    itemsList.Add(item5);
    itemsList.Add(item6);

    tables.Add(table1);
    tables.Add(table2);
    tables.Add(table3);
    tables.Add(table4);
    tables.Add(table5);
}
```

- GetTables(), GetItems() that returns all the tables and all the items initialized in the Management constructor.
- GetOrders[state](int tabID) that returns the orders, of the table with the id equals to tabID, with the state mentioned in the name of the function.
- InsertOrder(int tabId, list<Item> items) this function, receives a list of items and creates one or two orders, separating order to go to the kitchen, with the items of type kitchen, or to the bar with the items of type bar.
- PayTable(int tabId), this function, resets the table to the original state, without orders, but also writes in a file the table, and the total price paid.
- UpdateOrder[state](int orId) doesn't return anything, but changes the order with the orId to the state mentioned in the name of the method.

2.4 Events

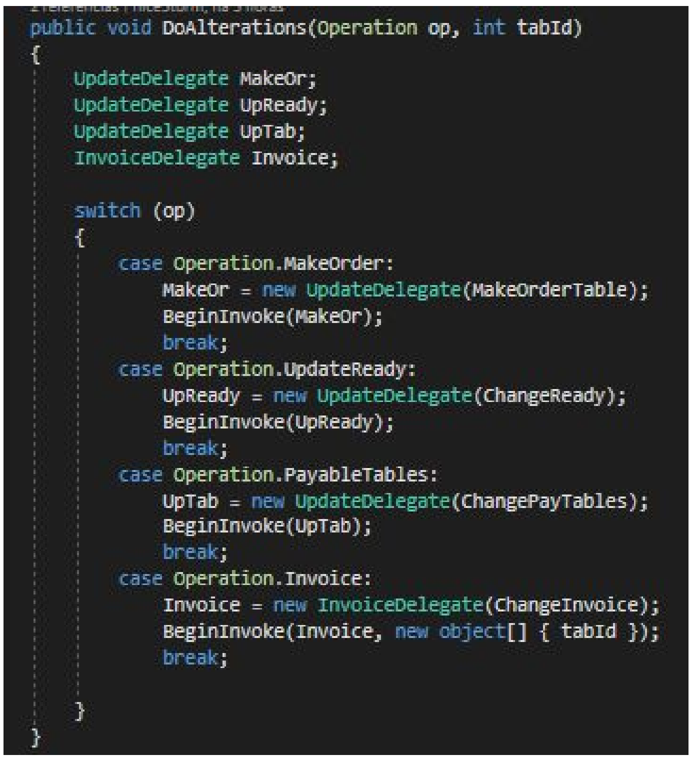
Events are call whenever the server needs to send a warning to the clients to update their information. Only some methods call that notification.

For example when a order is delivered, stops being Ready and changes to Done, and that send several notifications to the client:



```
public void UpdateOrderToDone(int orderId)
{
    Console.WriteLine("Delivering");
    foreach (Table t in tables)
    {
        foreach (Order o in t.Orders)
        {
            if (o.Id == orderId)
                o.UpdatetoDone();
        }
    }

    NotifyClients(Operation.UpdateReady, 1);
    NotifyClients(Operation.PayableTables, 1);
}
```

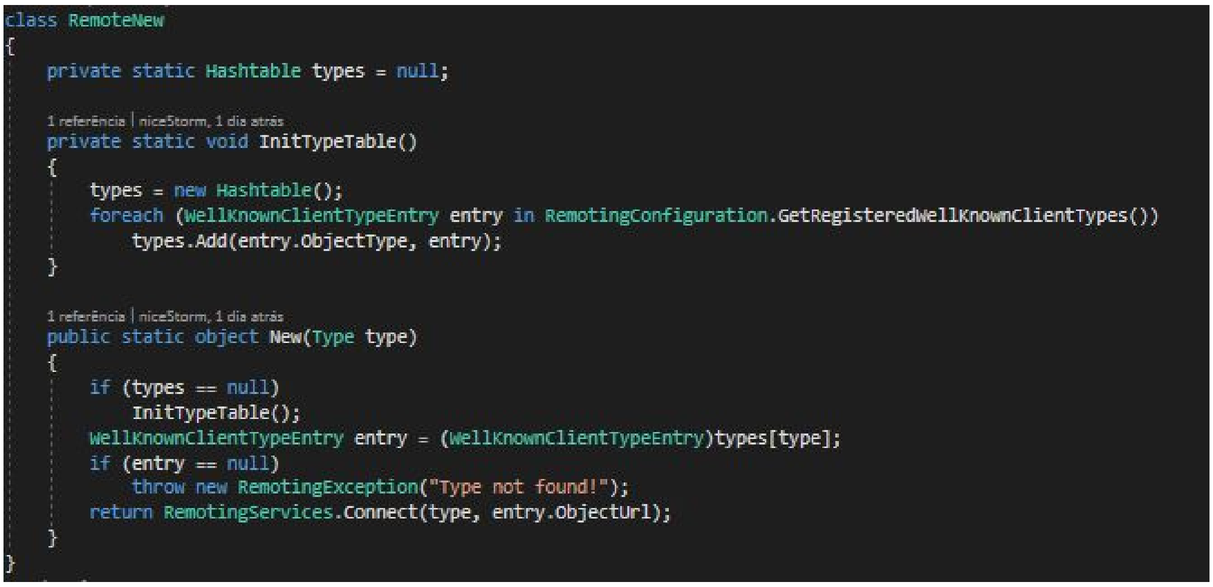
The notification is then dealt in the client:



```
public void DoAlterations(Operation op, int tabId)
{
    UpdateDelegate MakeOr;
    UpdateDelegate UpReady;
    UpdateDelegate UpTab;
    InvoiceDelegate Invoice;

    switch (op)
    {
        case Operation.MakeOrder:
            MakeOr = new UpdateDelegate(MakeOrderTable);
            BeginInvoke(MakeOr);
            break;
        case Operation.UpdateReady:
            UpReady = new UpdateDelegate(ChangeReady);
            BeginInvoke(UpReady);
            break;
        case Operation.PayableTables:
            UpTab = new UpdateDelegate(ChangePayTables);
            BeginInvoke(UpTab);
            break;
        case Operation.Invoice:
            Invoice = new InvoiceDelegate(ChangeInvoice);
            BeginInvoke(Invoice, new object[] { tabId });
            break;

    }
}
```

2.5 Subscribers

Each client connects to the remote object knowing only its interface, using the RemoteNew.New method

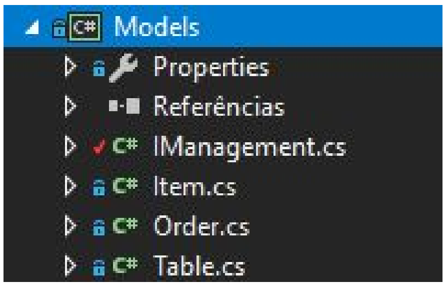


```
class RemoteNew
{
    private static Hashtable types = null;

    1 referência | niceStorm, 1 dia atrás
    private static void InitTypeTable()
    {
        types = new Hashtable();
        foreach (WellKnownClientTypeEntry entry in RemotingConfiguration.GetRegisteredWellKnownClientTypes())
            types.Add(entry.ObjectType, entry);
    }

    1 referência | niceStorm, 1 dia atrás
    public static object New(Type type)
    {
        if (types == null)
            InitTypeTable();
        WellKnownClientTypeEntry entry = (WellKnownClientTypeEntry)types[type];
        if (entry == null)
            throw new RemotingException("Type not found!");
        return RemotingServices.Connect(type, entry.ObjectUrl);
    }
}
```

3. Libraries

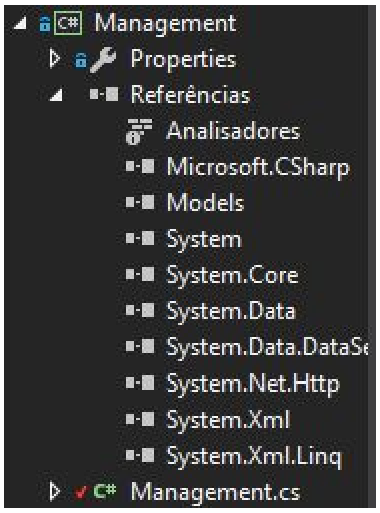
3.1 Models

The Models library contains all the data structures used in the terminals of the restaurant. This library contains the following data classes: Table, Order and Item, but also the interface IManagement through which the clients access information.



3.2 Management

The Management Library contains the methods used by the clients and references the Library Models.



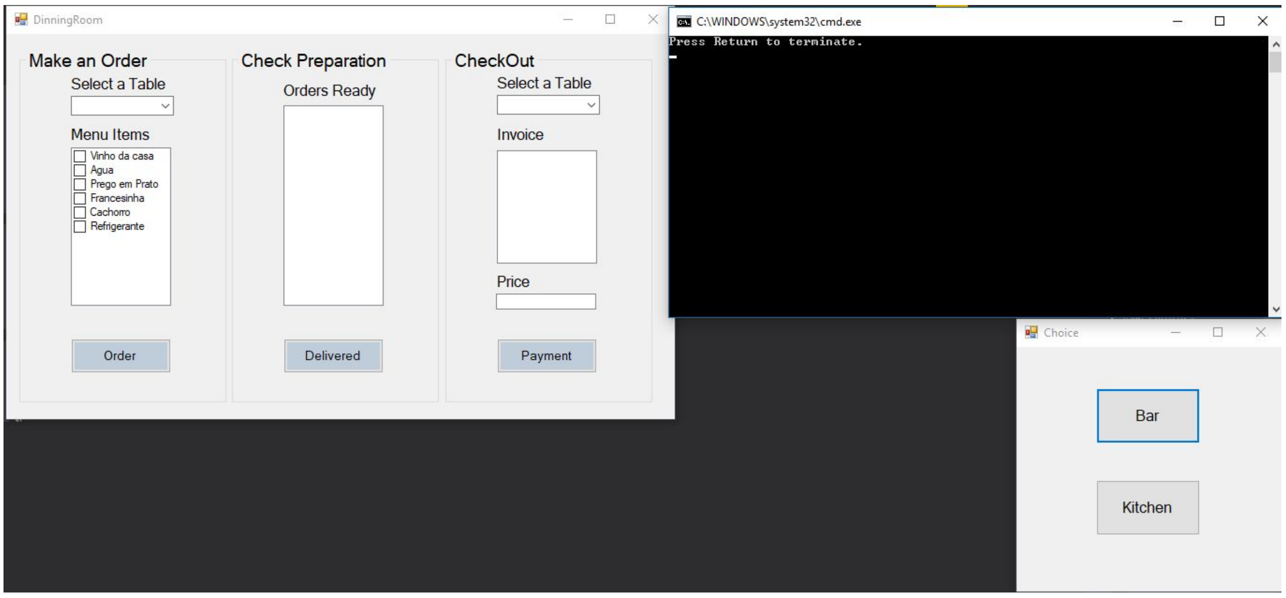
```
◢ Management
    ▷ Properties
    ◢ Referências
        Analisadores
        Microsoft.CSharp
        Models
        System
        System.Core
        System.Data
        System.Data.DataSe
        System.Net.Http
        System.Xml
        System.Xml.Linq
    ▷ Management.cs
```

4. Functionalities

- Create orders associated to a table in the dining room.
- Add to the orders already assigned to a table.
- Orders, when first created have the state “pending” and appear in the “pending orders” list in the kitchen or bar terminal.
- The orders change states, “pending” to “in preparation” to “ready” (when ready to be delivered) to “done” (when delivered).
- The tables can only be paid when all it’s orders are in the state “done”.
- When a table is chosen to be paid it’s shown the orders to be paid and the total price.

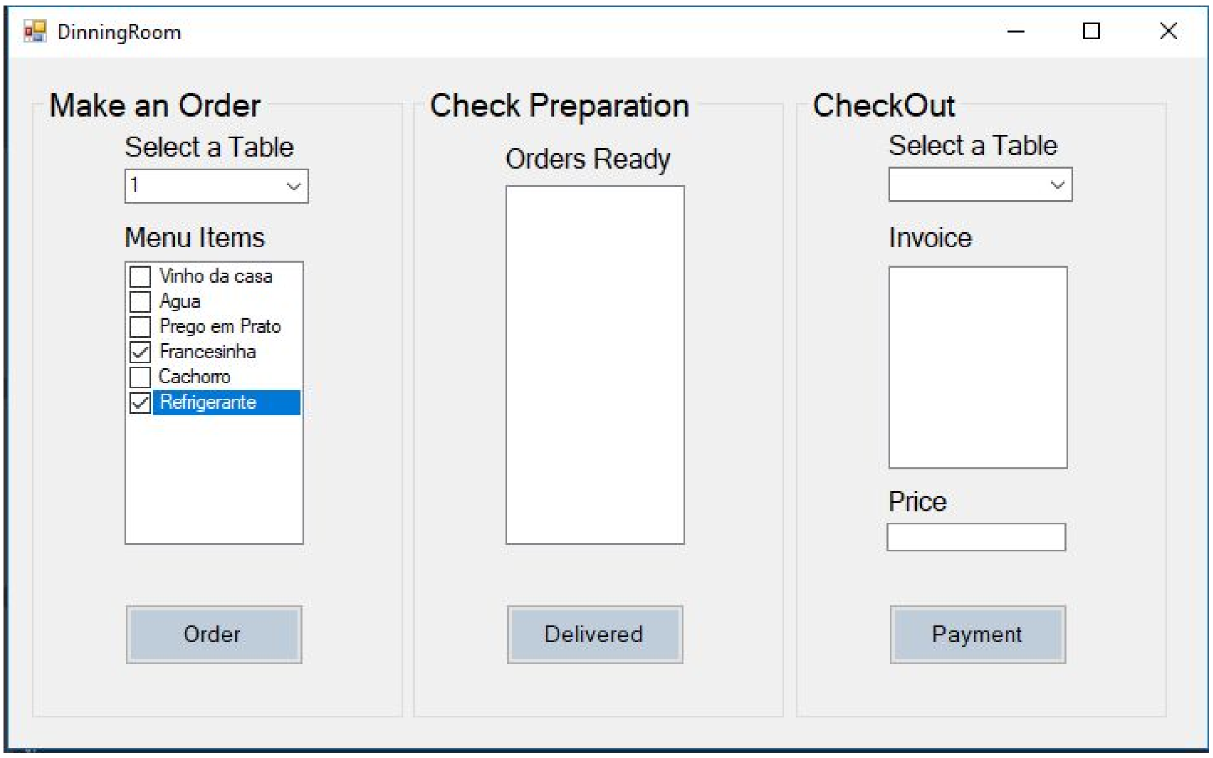
5. Sequences of use

5.1. Initial State

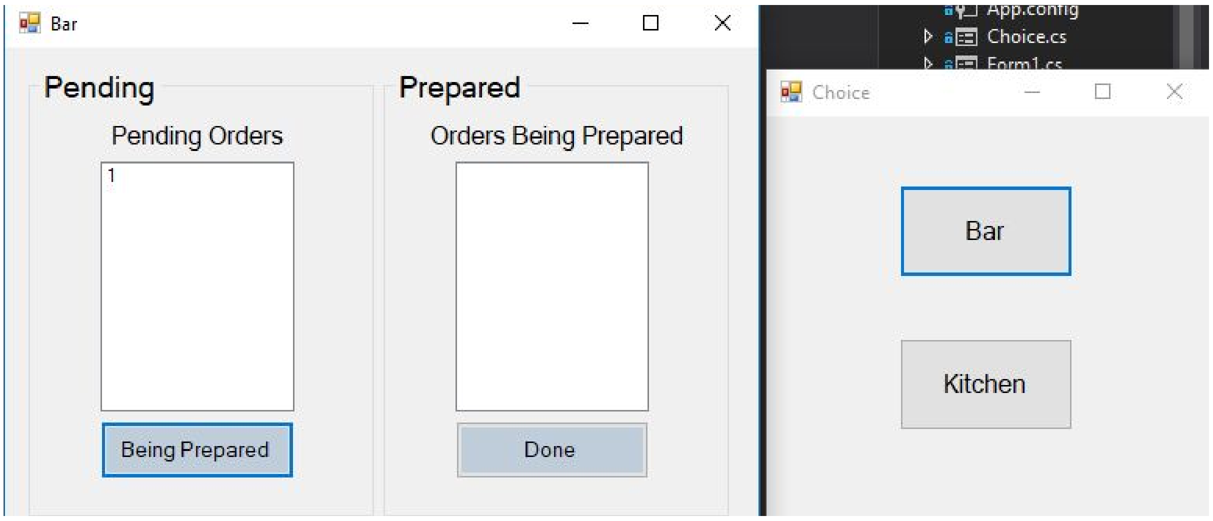


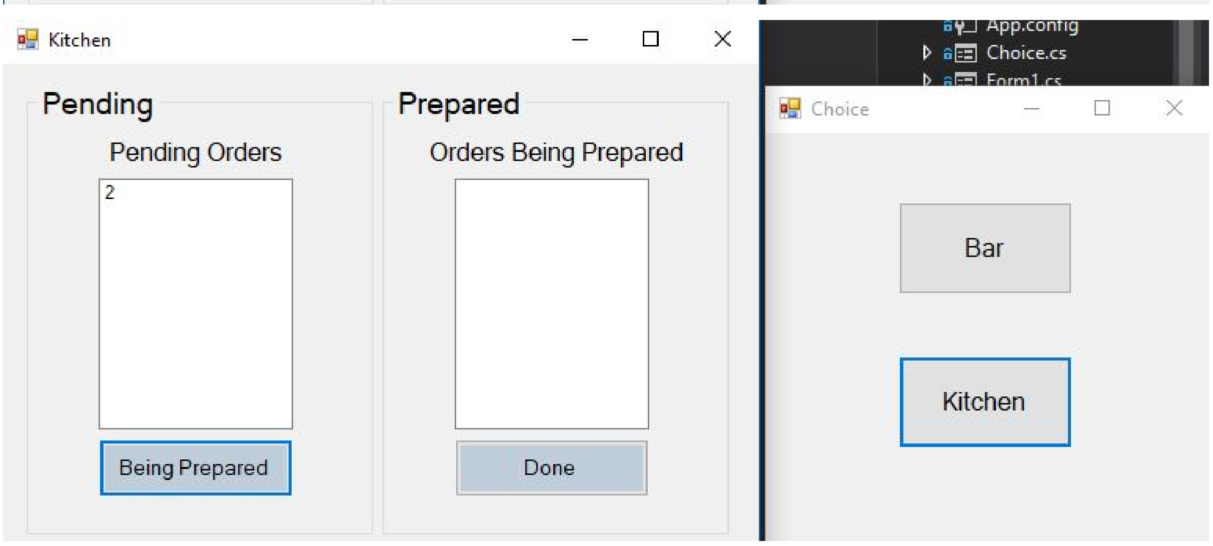
```
Press Return to terminate.
```

5.2. Making an Order

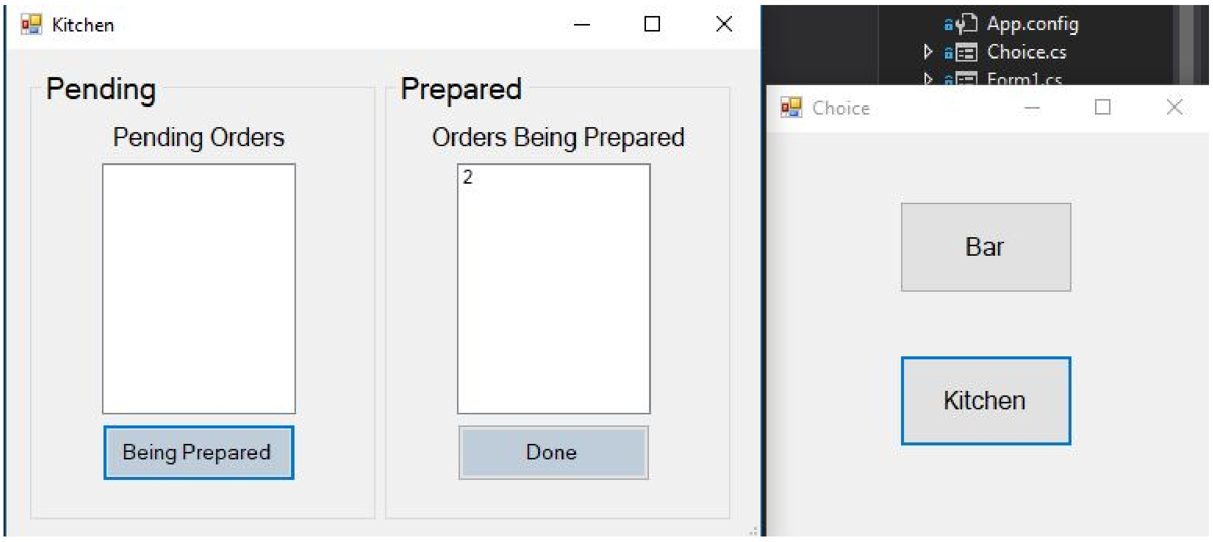


Since we made an order with items from both kitchen and bar, we divided that order in two.

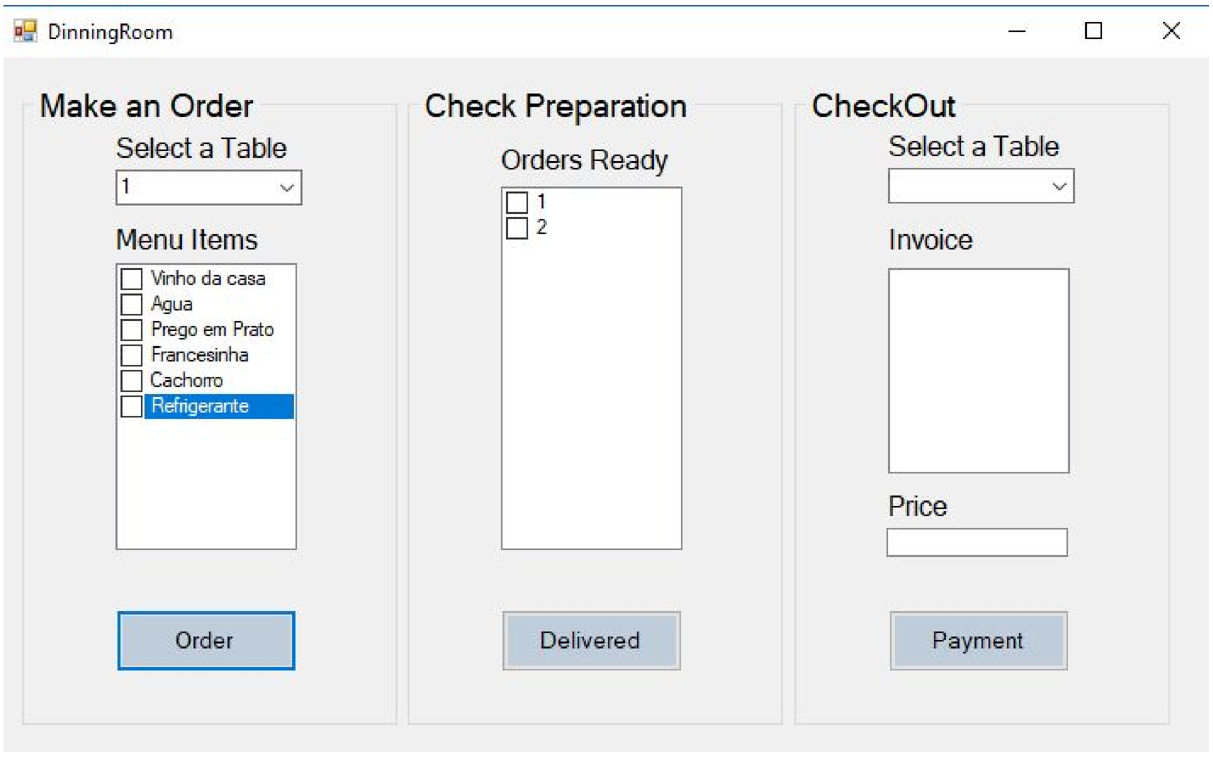




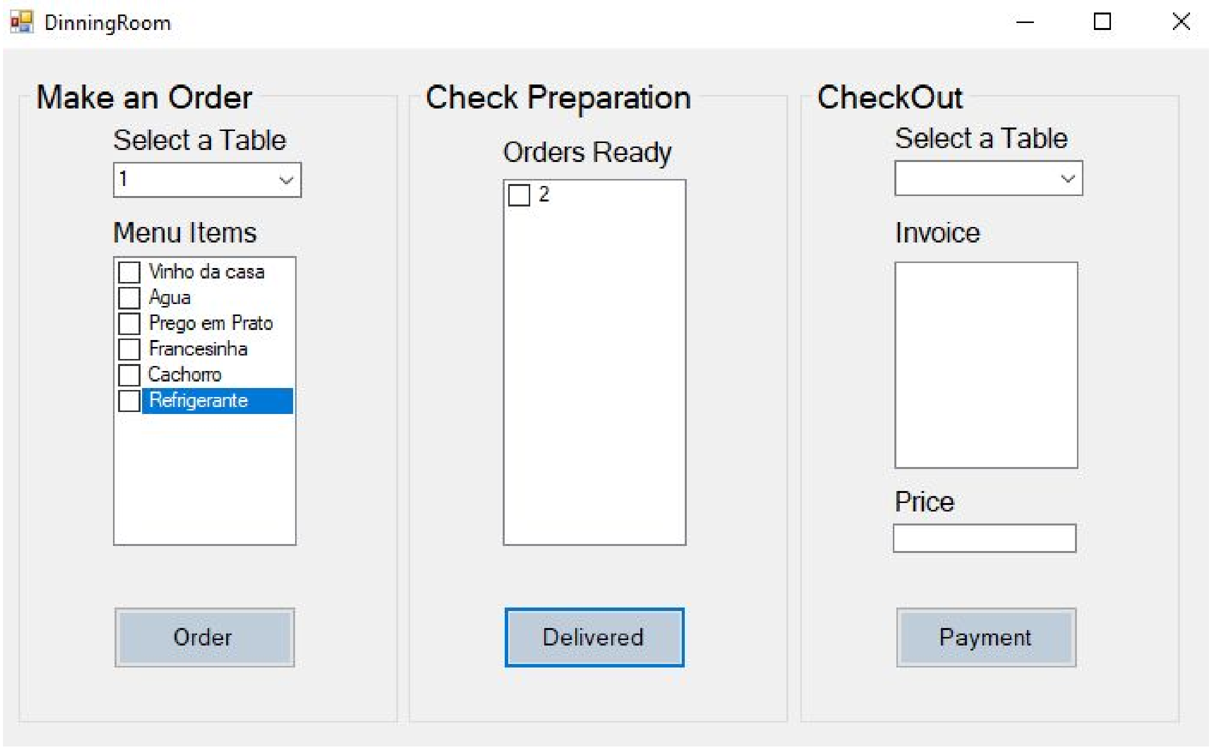
5.3. Changing the Order Status



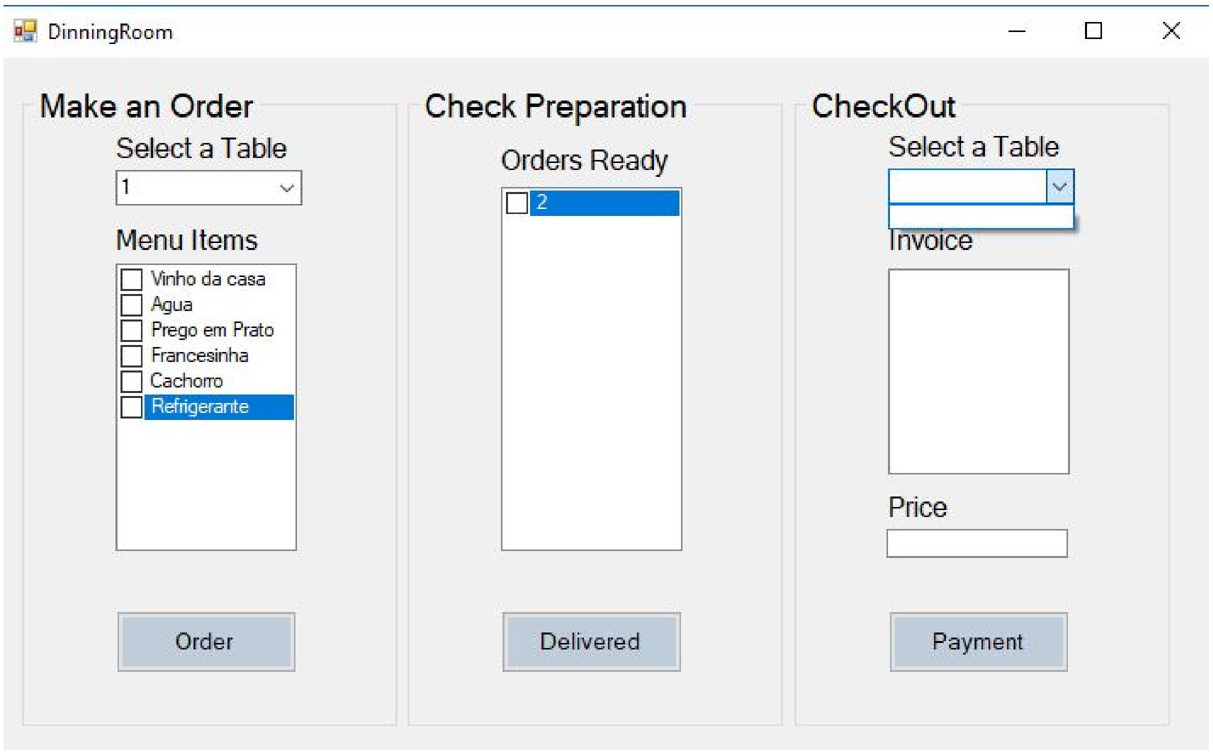
After changing the status of both orders we made, they appear in the dining room as Orders Ready.



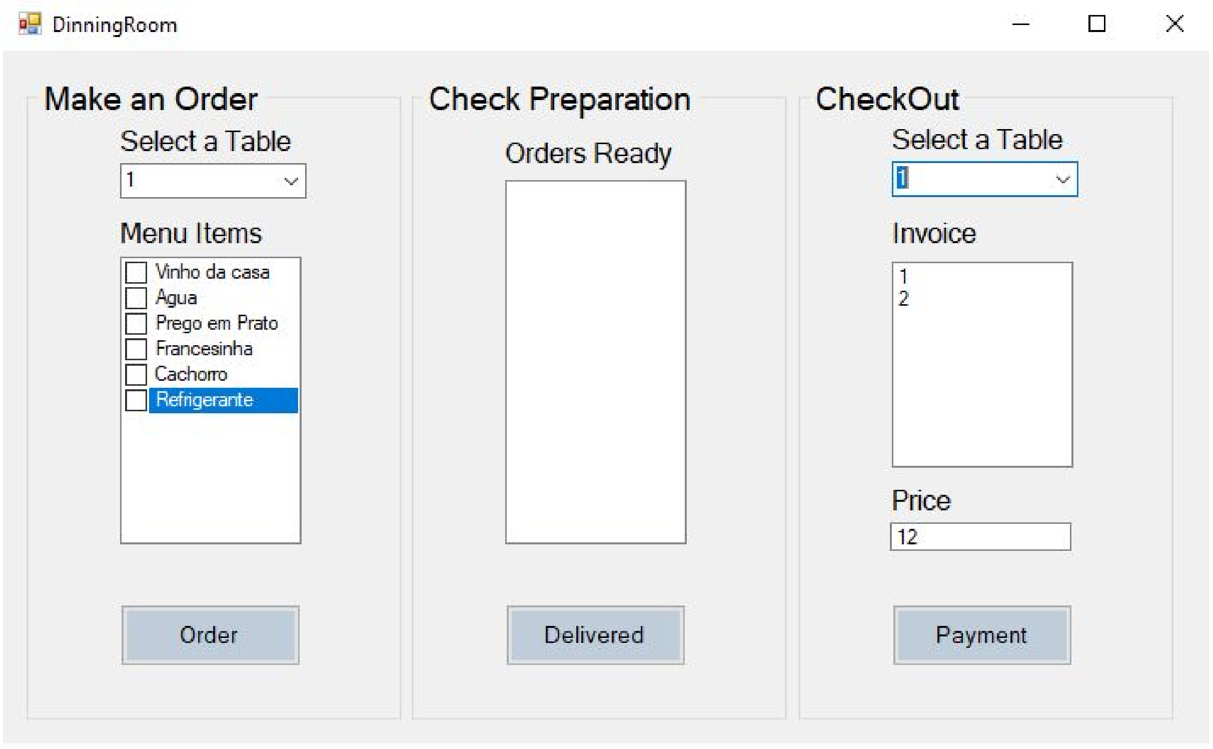
Then selecting the order we want to deliver, changes the status to Done.



5.3. Payment



As all orders of a table are done, the tables appears in the combobox, when selected the orders made show in the invoice and total price.


DinningRoom
Make an Order
Select a Table
1
Menu Items
Vinho da casa
Agua
Prego em Prato
Francesinha
Cachorro
Refrigerante
Order
Check Preparation
Orders Ready
Delivered
CheckOut
Select a Table
1
Invoice
1
2
Price
12
Payment