Lab 3: OORMS iteration 1 – taking orders

EEE320 - fall 2022

Introduction

The aim of this lab is to practice your understanding of use case descriptions, class diagrams, and sequence diagrams, and the translation of class and sequence diagrams into Python. To achieve this aim, you will implement an initial version of an Object-Oriented Restaurant Management System (OORMS) based on a design specification.

For this and subsequent labs we will be using the PyCharm IDE. You can install Python and PyCharm on your own machine by following the instructions provides in the Course Plan document provided in Moodle.

Submission

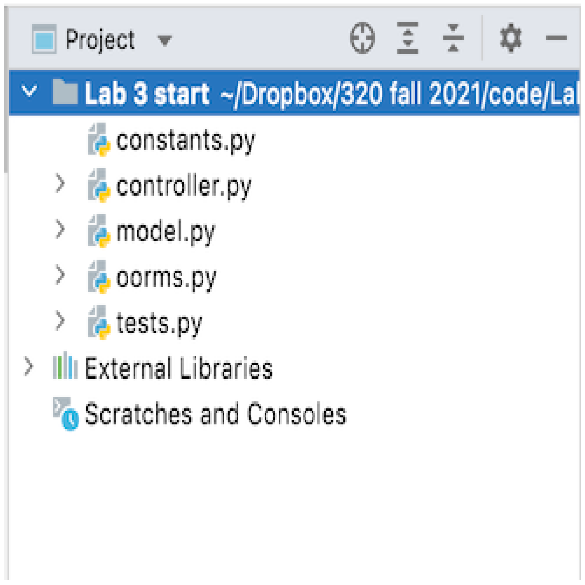
Your lab must submitted via the lab 3 Moodle assignment not later than 0800 hrs on Friday, 7 October 2022. The submission must consist of a lab report **in PDF form** and a copy of your source files, packaged together in a single zip file and named like `lab 3 Smith Jones.zip` where you replace "Smith" and "Jones" with the family names of your group members. **Do not** send a zip archive of your entire project: just the `.py` source files.

Your lab report must be well formatted, using the lab report provided on Moodle. Follow the instructions in the lab report template. You do not need to include copies of your code in the lab report itself.

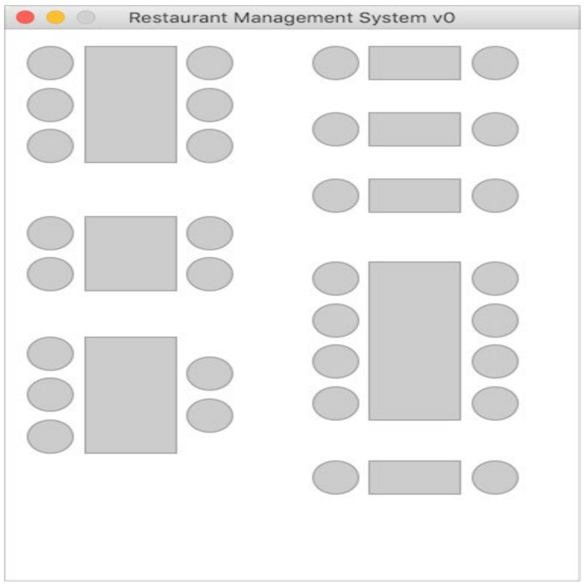
Setup

Launch PyCharm and create a new project, choosing a reasonable save location and project name. I recommend using the existing system interpreter (Python 3.10).

Once the main PyCharm window opens, download the file titled 'oorms-1-start.zip', provided with this lab, extract the contained files, and drag them onto your project. The project is the first line in PyCharm's project explorer; the left hand panel of the PyCharm interface. At this point, your project should look like what's shown below, with the `.py` files immediately below the project. If you've used a virtual environment, there will also be a `venv` folder — make sure you don't put your files inside the `venv`.



You should be able to run the oorms.py file by right clicking the filename and choosing *Run 'oorms'*. This should result in a window similar to the figure below.



You should also be able to run the unit tests in the tests.py file by right clicking on the filename and choosing *Run 'Python tests in test...'*. This should open a pane at the bottom of the PyCharm window showing that one test was run and passed.

Implementation

Starting from the provided code, extend OORMS so that it fully implements the specification below. The complete user interface code has been provided for you. We suggest you implement one sequence diagram at a time, in the order listed, using the class diagram for reference. *I.e.* do not start by attempting to implement the entire class diagram; implement just the classes needed for each sequence diagram as you go.

There is a test case in tests.py corresponding to each sequence diagram. Uncomment the test cases as you go and run them to ensure your code is working correctly. Reading the test cases may give you some hints about how the code needs to be implemented.

There are three TODO: notes in the code with instructions you will need to follow at appropriate points in your development: one in the oorms.py file and two in the model.py file.

Other than the TODO: mentioned above, you do not need to modify the oorms.py and constants.py files.

OORMS Iteration 1 Specification

Use case

Take a table's orders

Preconditions: The server is logged into the system and the restaurant view is visible. The server is ready to take the orders from a table.

1. The server selects the appropriate table from the tables in the restaurant.
2. The system presents a view of the table.
3. The server selects a seat at the table.
4. The system presents the current order for that seat. For a seat that has not yet ordered, this would be empty. The system also presents the available menu items.
5. The server selects an available menu item.
6. The system adds the selected item to the seat's order. *Steps 5 and 6 may be repeated any number of times.*
7. The server requests that the order be placed. Any items that have not already been ordered are marked as ordered. *(In a later iteration, these will be added to the kitchen's queue of orders to fill.)*
8. The system returns to the view of the table, as presented in step 2. *Steps 3 through 7 may be repeated any number of times.*
9. The server indicates that they are done with the current table.
10. The system returns to the restaurant view.

Alternates and exceptions:

Step 7. The server requests that the current order modification be cancelled. Any items already marked as orders are unaffected, but any items not yet ordered are removed.

(More alternates and exceptions will be added in a future iteration)

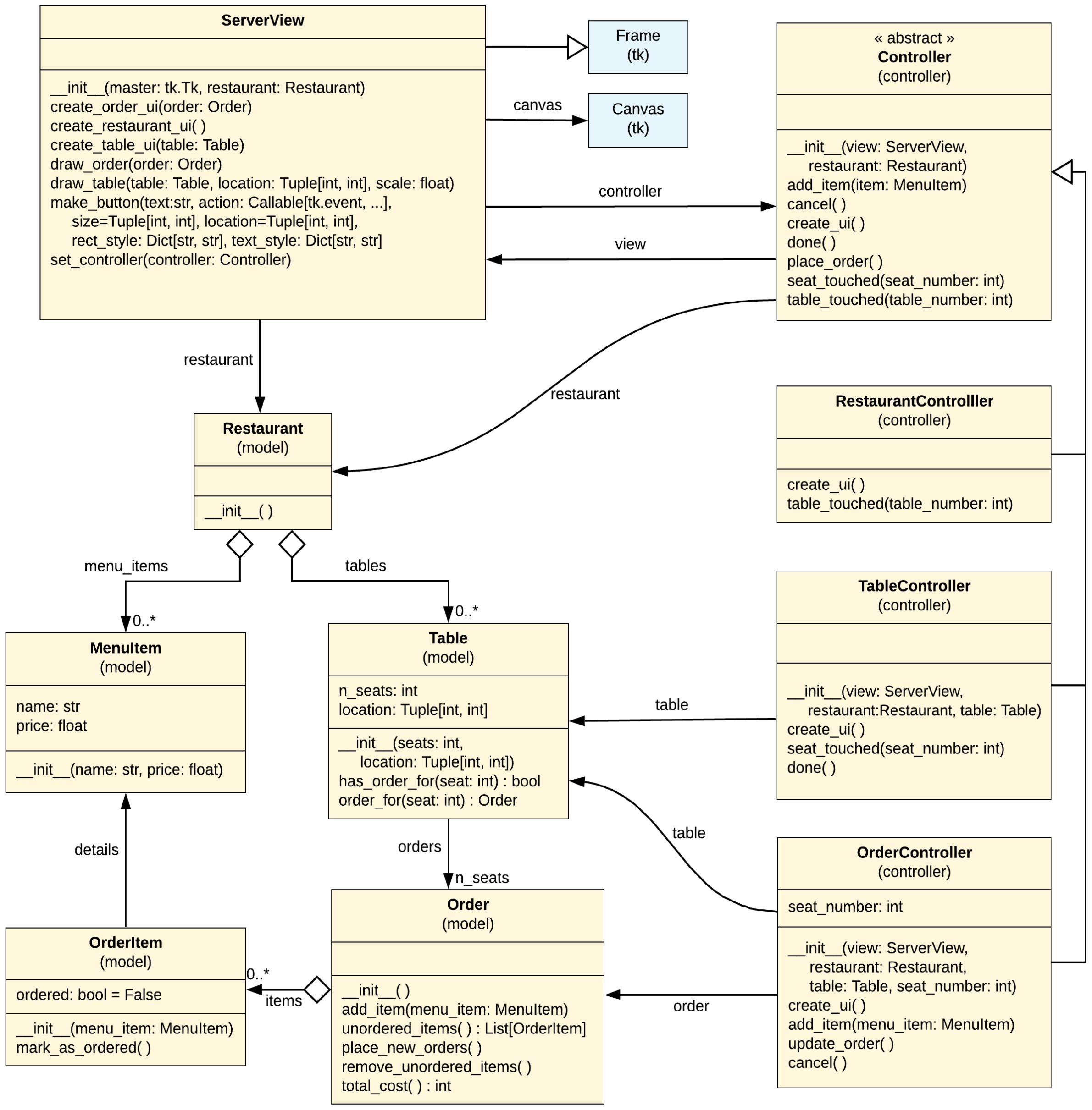
User interface notes

On the user interface, seats that have current orders associated with them are coloured green; seats that have no current orders are coloured grey. This is implemented by the code in oorms.py below the TODO: at line 68.

With this lab assignment, you can find a video of the required behaviour.

Class diagram

A class diagram of the system is below. For this and all sequence diagrams below, we provide you the *PDF version* on Moodle.



ServerView
__init__(master: tk.Tk, restaurant: Restaurant)
create_order_ui(order: Order)
create_restaurant_ui( )
create_table_ui(table: Table)
draw_order(order: Order)
draw_table(table: Table, location: Tuple[int, int], scale: float)
make_button(text:str, action: Callable[tk.event, ...], size=Tuple[int, int], location=Tuple[int, int], rect_style: Dict[str, str], text_style: Dict[str, str]
set_controller(controller: Controller)
Frame (tk)
canvas
Canvas (tk)
« abstract » Controller (controller)
__init__(view: ServerView, restaurant: Restaurant)
add_item(item: MenuItem)
cancel( )
create_ui( )
done( )
place_order( )
seat_touched(seat_number: int)
table_touched(table_number: int)
controller
view
restaurant
restaurant
Restaurant (model)
__init__( )
RestaurantControlller (controller)
create_ui( )
table_touched(table_number: int)
menu_items
tables
0..*
0..*
MenuItem (model)
name: str
price: float
__init__(name: str, price: float)
Table (model)
n_seats: int
location: Tuple[int, int]
__init__(seats: int, location: Tuple[int, int])
has_order_for(seat: int) : bool
order_for(seat: int) : Order
TableController (controller)
__init__(view: ServerView, restaurant:Restaurant, table: Table)
create_ui( )
seat_touched(seat_number: int)
done( )
table
table
details
orders
n_seats
Order (model)
__init__( )
add_item(menu_item: MenuItem)
unordered_items( ) : List[OrderItem]
place_new_orders( )
remove_unordered_items( )
total_cost( ) : int
OrderController (controller)
seat_number: int
__init__(view: ServerView, restaurant: Restaurant, table: Table, seat_number: int)
create_ui( )
add_item(menu_item: MenuItem)
update_order( )
cancel( )
order
OrderItem (model)
ordered: bool = False
__init__(menu_item: MenuItem)
mark_as_ordered( )
0..*
items

Sequence Diagrams

Restaurant view: Server selects a table

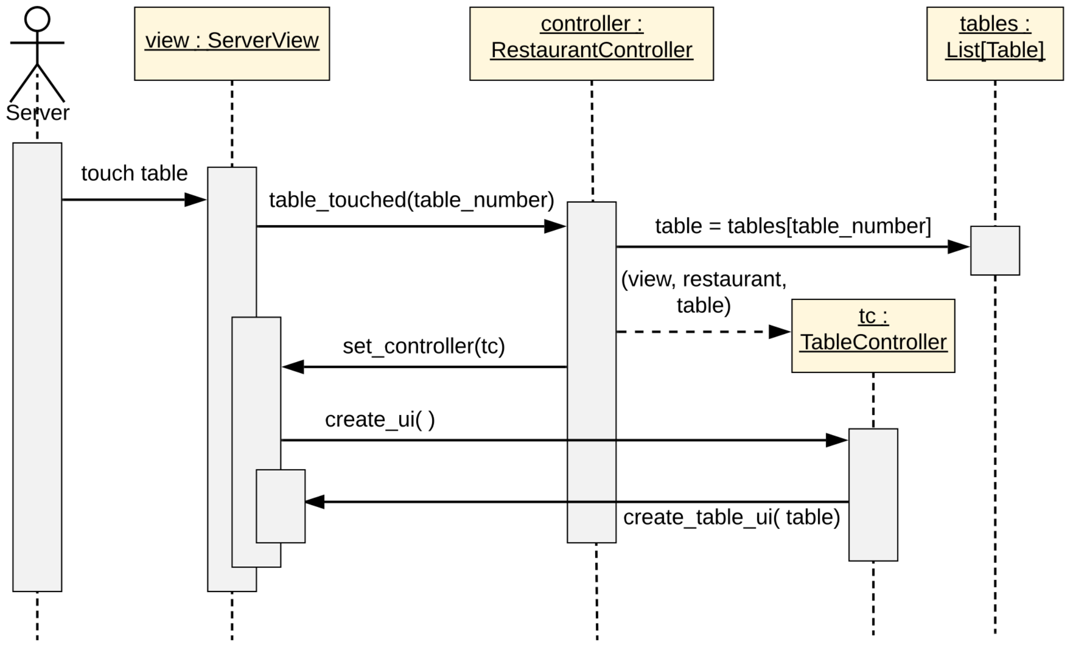


Table view: Server presses the Done button

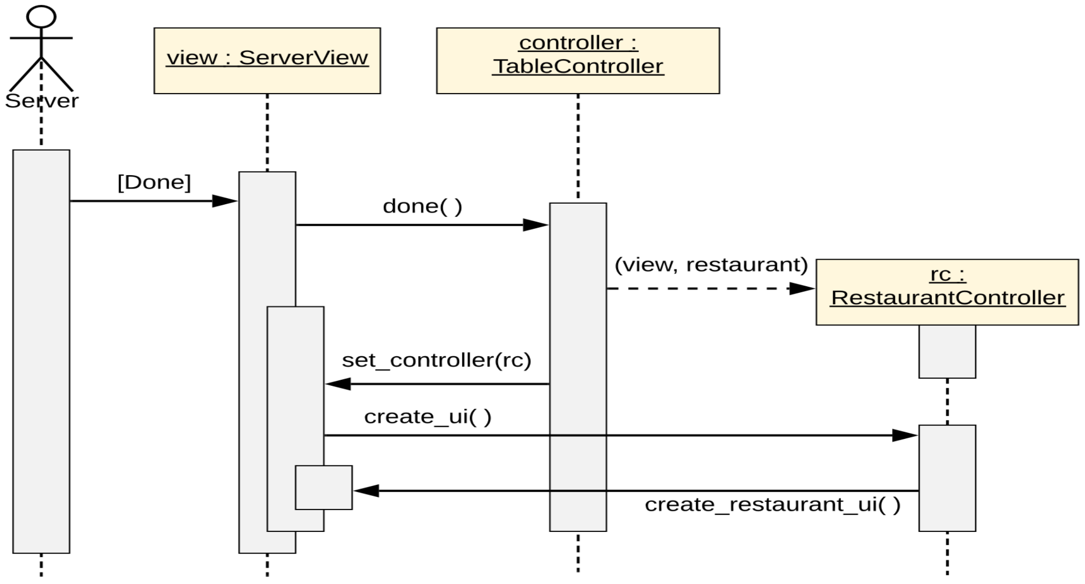
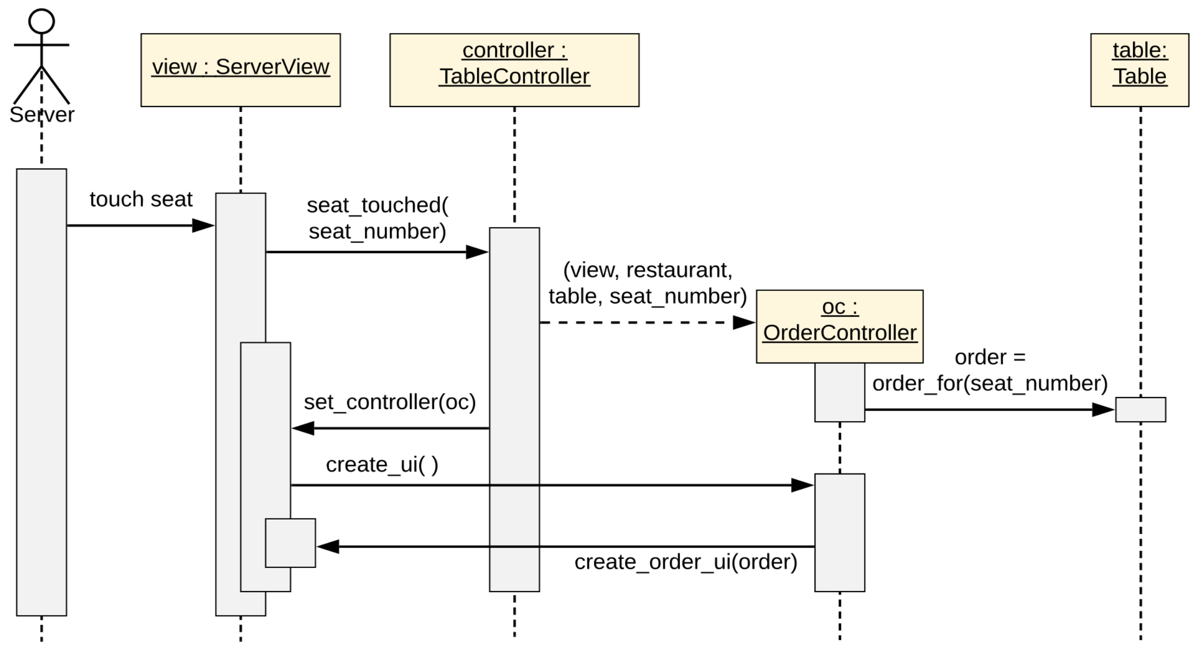
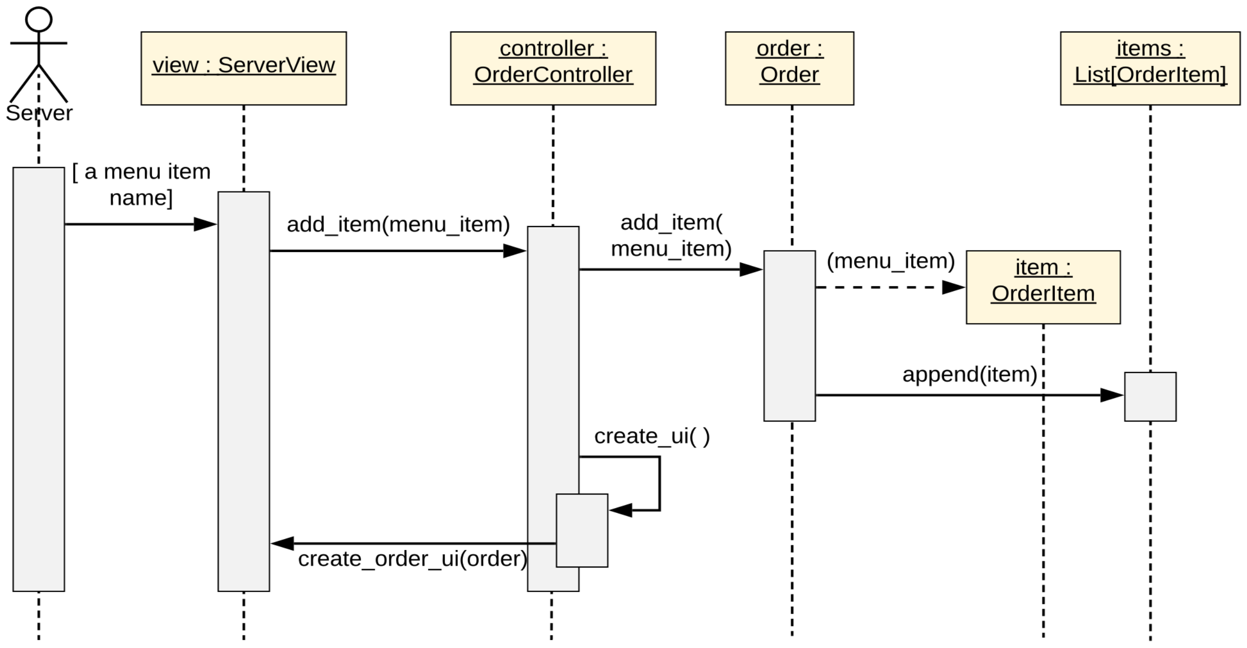


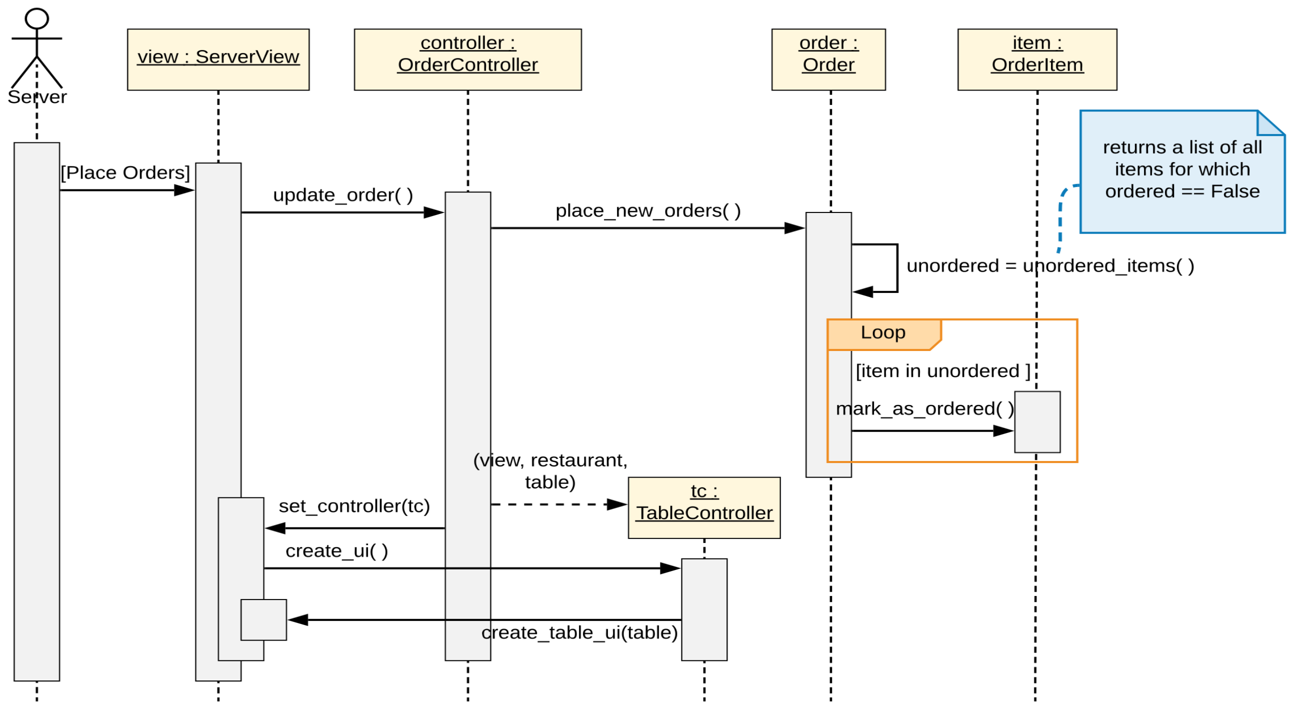
Table view: Server selects a seat


Server
view : ServerView
controller : TableController
table: Table
touch seat
seat_touched(seat_number)
(view, restaurant, table, seat_number)
oc : OrderController
order = order_for(seat_number)
set_controller(oc)
create_ui( )
create_order_ui(order)

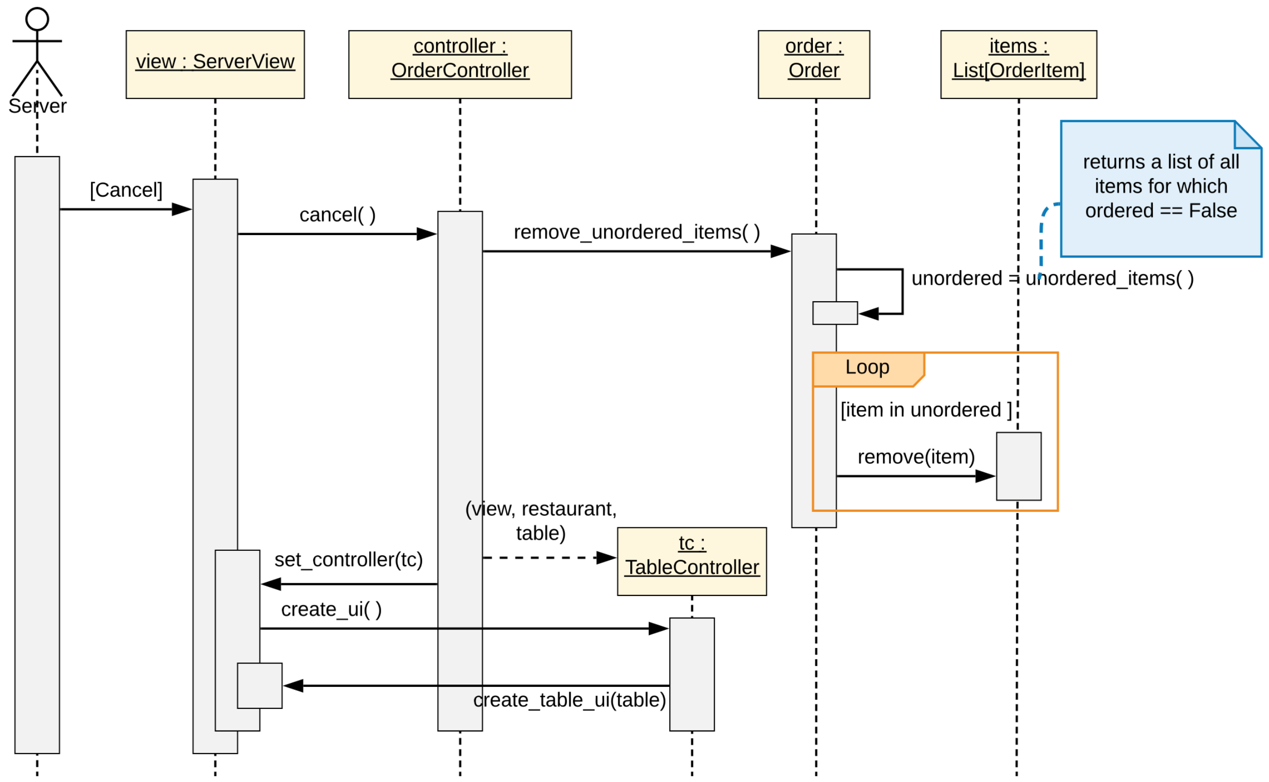
Order view: Server selects a menu item to be ordered



Order view: Server presses the Place Orders button



Order view: Server presses the Cancel button


Server
view : ServerView
controller : OrderController
order : Order
items : List[OrderItem]
returns a list of all items for which ordered == False
[Cancel]
cancel( )
remove_unordered_items( )
unordered = unordered_items( )
Loop
[item in unordered]
remove(item)
(view, restaurant, table)
tc : TableController
set_controller(tc)
create_ui( )
create_table_ui(table)