SOM-ITSOLUTIONS

Java

Designing of an OOAD system from scratch using UML

SOMENATH MUKHOPADHYAY

som-itsolutions
#A2 1/13 South Purbachal Hospital Road Kolkata 700078 Mob: +91 9748185282
Email: som@som-itsolutions.com / som.mukhopadhyay@gmail.com
Website: http://www.som-itsolutions.com/
Blog: www.som-itsolutions.blogspot.com

From the last few days i have been thinking to share my idea about how to design software system from scratch with the help of object oriented principles. The way the OOPS is taught to the students may enable them to write small programming assignments, but when it comes to design a whole system, many of them may falter. I would like to address that part with the basic principle of OOAD and tell you how to traverse from the problem domain to the solution domain with the help of OOAD.

So, lets start. I have thought of an example to design a Restaurant system from scratch. I have used Java as the language, but one can use any OO language to achieve it.

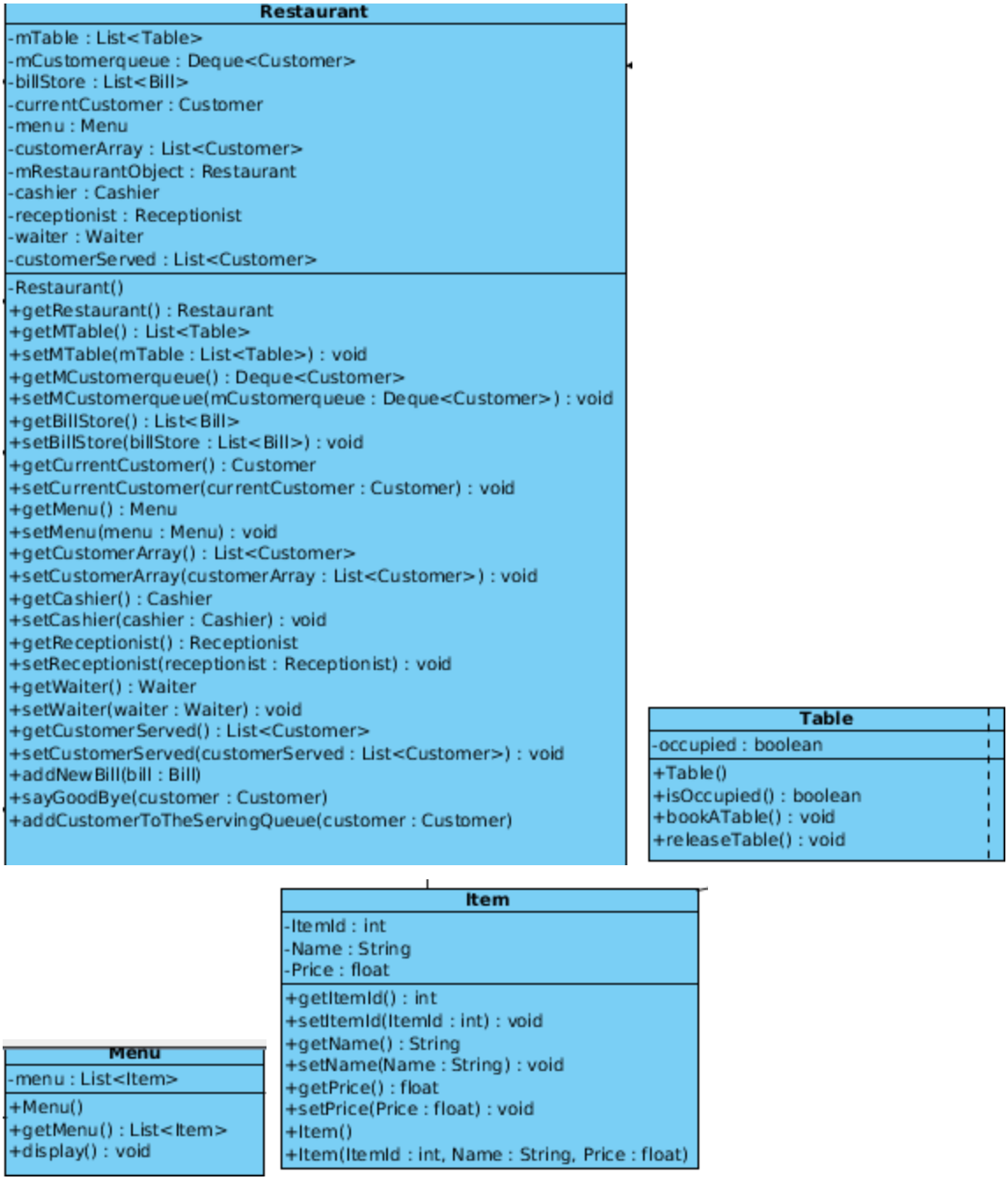
To start designing of a software system using OOAD, one has to first identify the main entities participating in that system. So, in a Restaurant system who are the main participants? You guessed it right. A restaurant itself. Then the next entity is of course a customer. I think these are the main two entities. Now what does a restaurant consist of. It consists of some tables, a menu card, and food items. Another thing i have forgotten to mention. A restaurant should have a FIFO queue for its customers… right?

Now what kind of operation a Restaurant has to do? It should have some functionality to know if all tables are occupied, and if not which one is vacant; it should be able to book a vacant table when a new customer arrives. And when the customer leaves, it should be able to release that table and if anybody is waiting in its FIFO queue, it should be able to allocate that vacant table to that customer. It should also keep a track about who is the current customer it is serving and which customer has been allocated which table. Then it should also be able to generate a bill.

Now what does a Menu consist of? It has a list of some food items… right? So from Menu's angle, it needs an entity called Item. Now what are the attributes of a food item vis-a-vis a restaurant? Each item should have an ID, a Name and its Price… right?

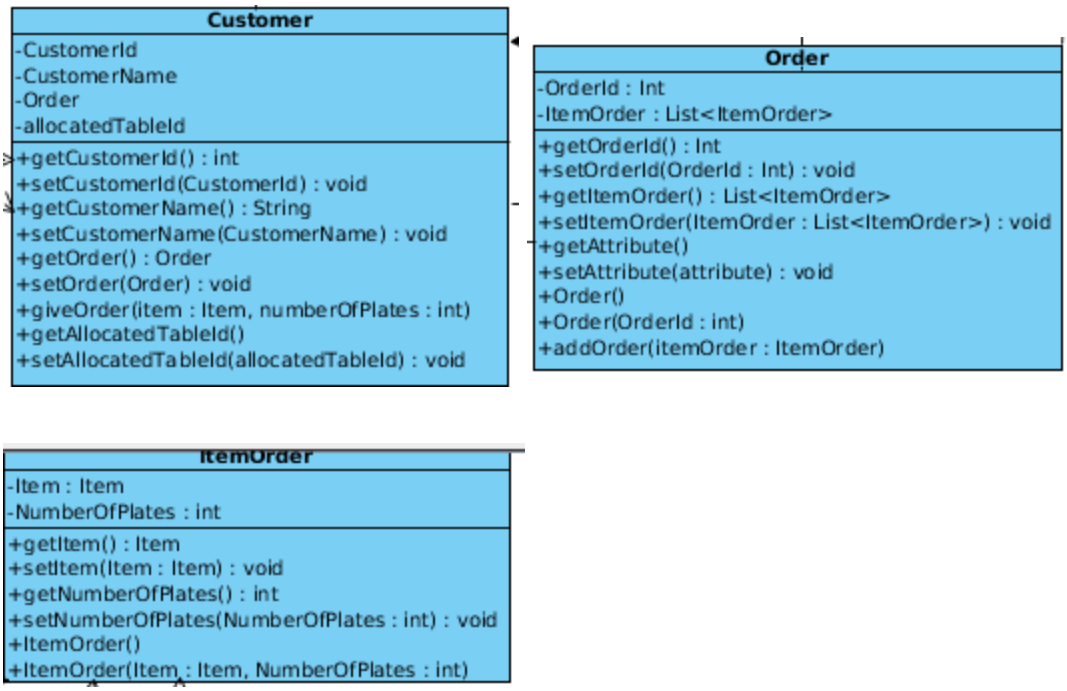
Now think that a restaurant will have a list of tables scattered systematically. So what kind of attributes a Table in a Restaurant may have? It should have an ID and it should know whether it is occupied or not. What kind of operation does a Table need? When a customer leaves the table, it should be able to tell the outside that it is occupied… right? On the other hand when a customer leaves the table it should let the outside world know that it is vacant and ready to accept new customer.

With these things in mind let us design the Restaurant, Table, Menu and Item classes.



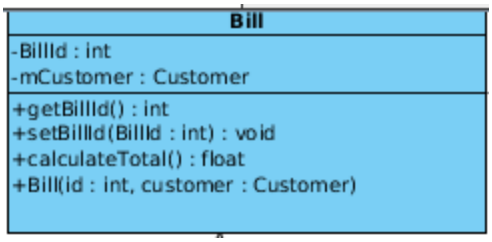
Now let us talk about the other main entity called the Customer. What are the attributes of a customer should have when he enters a restaurant. he must have an ID. i am not talking about his passport or SSN or Aadhar card ID. This ID is with respect to the restaurant by which the customer can be identified within the Restaurant premise. We will also add Name as another attribute to a

customer. Although there is no absolute need for this from the OOAD designer's perspective, this is needed only for decoration as you will see later. And what kind of operation a customer usually does. He looks at the menu, decides what he wants to have and how many plates… right? So the customer entity should have an operation called giveOrder. Now think about this. In order to give order customer needs an entity which will hold the customer's choice, i.e., which item and how many plates for that item. Lets call this entity as ItemOrder, each ItemOrder object representing a specific Item and its number of plates ordered by the customer. So in the actual order by a customer, there will be a collection for all these ItemOrders objects created by the customer. Lets create a class as Order which will hold a list of such ItemOrder objects. This Order class can be used by the Resaturant entity for calculating the total amount to be billed to the customer… right? So in essence, we have got three major classes to design the customer and his flow of orders, namely Customer, ItemOrder, Order. Lets see how these classes will look like.

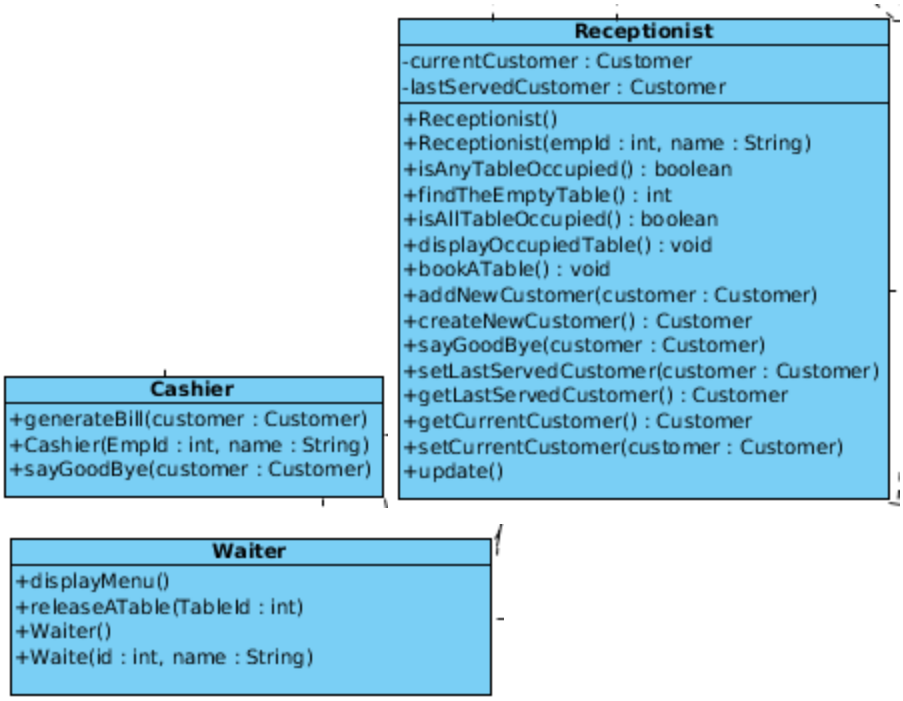


Remember, all setters and getters functions have been removed from these diagrams (except ItemOrder) to make it simple.

Okay, another helper class we need. That is the bill class. It should have the functionality to calculate the total amount to be billed for a particular customer.



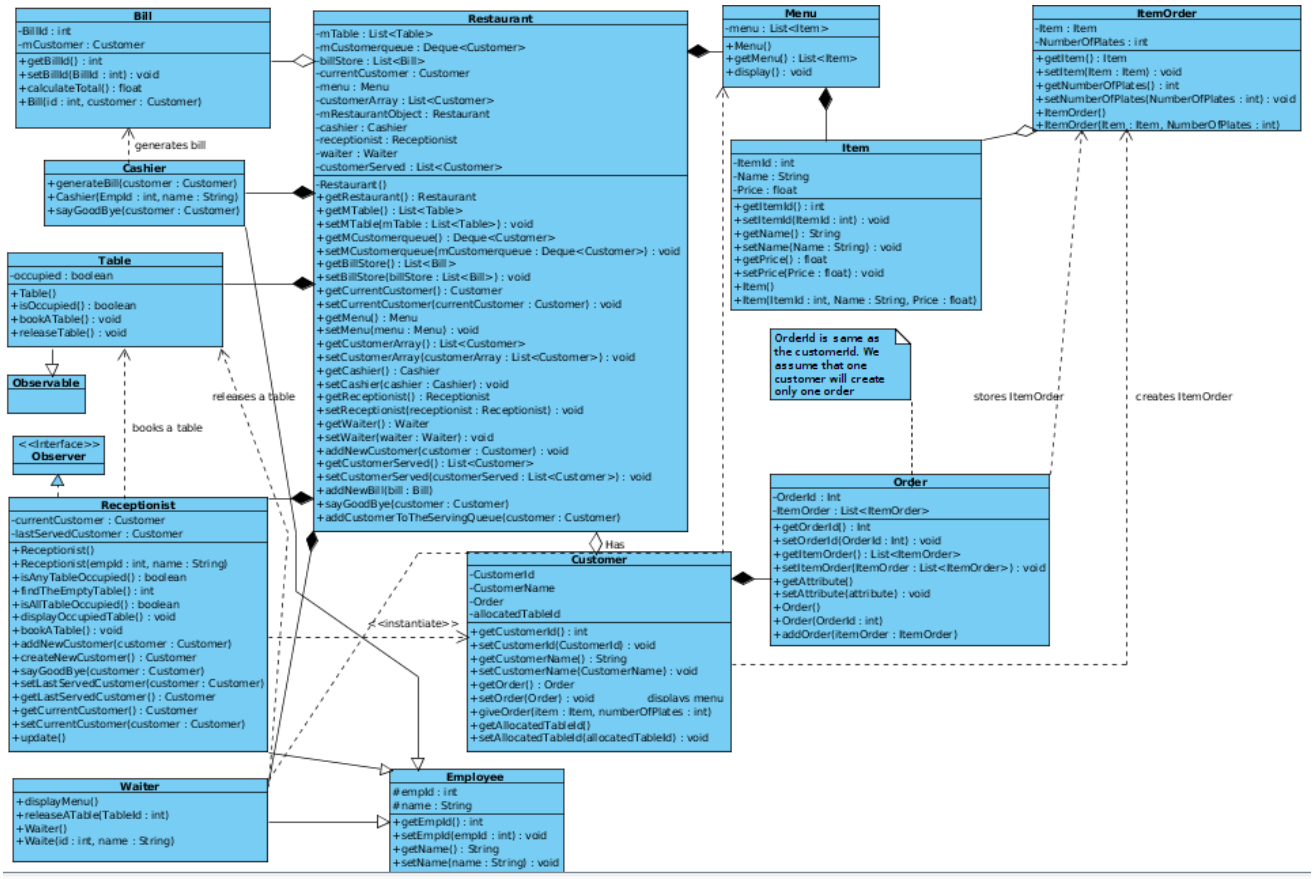
And in a Restaurant, the other main entities are Cashier, Receptionist and Waiter.



With all these classes ready, let us design the complete class diagram of the Restaurant System.

As you have probably guessed that a Restaurant ***HAS*** Tables and Restaurant **HAS** Menu. Menu ***HAS*** Item (rather items, a list of item). Restaurant has an ***ASSOCIATION*** with the Customer. The Bill stores a reference to a Customer for future use. Hence Bill has an ***AGGREGATION*** relationship with Customer. Each Customer creates an Order. Hence Customer ***HAS*** Order. And an Order holds a list of ItemOrders each representing and amount ordered for a particular Item. Hence Order ***HAS*** ItemOrder. Each ItemOrder keeps a reference of Item. Hence ItemOrder has an ***AGGREGATION*** relationship with Item. The Restaurant ***USEs*** Bill to calculate the total bill.

And with this explanation, the complete class diagram will look as the following.



Please note that the complete signatures of the operations are not given in the UML diagrams to make it simple. You can refer the respective class from the source code attached to have an idea of each operation.

Let me explain the UML in more detail. As you can see that Restaurant has a one-to-one HAS relationship with Menu and an one-to-many HAS relationship with the TABLE. This is quite obvious… right? Because a single Restaurant will have only one Menu list and it may have one to many number of TABLEs. The relationship of the Restaurant with the Customer is Association. Because a customer is an Independent entity. The Restaurant simply adds the new customer to its FIFO queue. As i have already mentioned, that a Customer HAS a Order (one-to-one relationship). Because whenever a customer plans to order, he needs to create an Order which he will populate with different ItemOrders(remember ItemOrder keeps a reference to an Item and how many plates the customer needs). As the ItemOrder keeps a reference to the Items, it obviously has an Aggregation relationship with the Item. A bill also has to keep a reference of the customer because it needs to identify for which customer the Bill has been generated. Hence the Bill also has an Aggregation relationship with Customer. The Restaurant has a USE relationship with the Bill because it creates the Bill object as a local variable in the generateBill function. Now probably you have acquired some sorts of idea about how this whole system has been designed using UML and OOAD.

The code for this whole Restaurant System has been given below. It is, of course in accordance to this class diagram given earlier.

The Restaurant Class:

```
package com.somitsolutions.training.java.restaurant;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.Iterator;
import java.util.List;
import java.util.Observer;




public class Restaurant {
	public static final int MAX_NUMBER_OF_TABLES = 2;
	private List<Table> mTables;
	//whenever a new customer comes we add him to the front of this queue
	private Deque<Customer> mCustomerQueue;
	private List<Customer> customerServed;
	private Menu mMenu;
	private List<Bill> billStore;
	private static Restaurant mResturantObject = null;
	//private Customer currentCustomer;
	private List<Customer> customerArray;
	Employee cashier;
	Employee receptionist;
	Employee waiter;

	private Restaurant(){
		receptionist = new Receptionist(1, "Rita");
		waiter = new Waiter(10, "Shyam");
		cashier = new Cashier(100, "Ram");

		mTables = new ArrayList<Table>();
		mMenu = new Menu();
		billStore = new ArrayList<Bill>();
		mCustomerQueue = new ArrayDeque<Customer>(5);
		customerArray = new ArrayList<Customer>();
		customerServed = new ArrayList<Customer>();
		//currentCustomer = null;
		for(int i = 0; i < MAX_NUMBER_OF_TABLES; i++){
```

```
                    Table t = new Table();
                    t.addObserver((Observer) receptionist);
                    mTables.add(t);
              }
       }

       public List<Table> getTables(){
              return mTables;
       }

       public Deque<Customer> getCustomerQueue() {
              return mCustomerQueue;
       }

       public void setCustomerQueue(Deque<Customer> customerQueue) {
              this.mCustomerQueue = customerQueue;
       }

       public Menu getMenu() {
              return mMenu;
       }

       public void setMenu(Menu mMenu) {
              this.mMenu = mMenu;
       }

       //Singleton Pattern
       public static Restaurant getRestaurant(){
              if(mResturantObject == null)
                    mResturantObject = new Restaurant();
              return mResturantObject;
       }

       public Customer getCustomer(int customerId){
              Iterator it = (Iterator) customerArray.iterator();

              while(it.hasNext()){
                    Customer c = (Customer) it.next();
                    if(customerId == c.getCustomerId()){
                          return c;
                    }
              }
              return null;
       }
```

```
	public List<Bill> getBillStore(){
		return billStore;
	}

	public Cashier getCashier(){
		return (Cashier)cashier;
	}

	public List<Customer> getCustomerArray(){
		return customerArray;
	}

	public Receptionist getReceptionist(){
		return (Receptionist)receptionist;
	}

	public Waiter getWaiter(){
		return (Waiter)waiter;
	}

	public void addCustomerToTheServingArray(Customer customer){
		customerArray.add(customer);
	}
	public void bookTable(int tableNumber){
		mTables.get(tableNumber).bookTable();
	}

	public void releaseTable(int tableNumber){
		mTables.get(tableNumber).releaseTable();
	}

	public void addNewBill(Bill bill){
		billStore.add(bill);
	}

	public List<Customer> getCustomerServed() {
		return customerServed;
	}

	public void setCustomerServed(List<Customer> customerServed) {
		this.customerServed = customerServed;
	}

	public void sayGoodBye(Customer customer){
```

```
            customerArray.remove(customer);
      }
}
```

The Customer Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.Random;

public class Customer{
      private int mCustomerId;
      private Order mOrder;
      private String mCustomerName;
      private int allocatedTableId;


      public Customer(){
            Random r = new Random();
            mCustomerId = r.nextInt(10000); // A random number between 0 & 10000
            //mOrder = new Order(mCustomerId);
            //addObserver(Restaurant.getRestaurant());
      }

      /*public Customer(int customerId){
            //mCustomerId = customerId;
            mOrder = new Order(customerId);
      }*/

      public int getCustomerId() {
            return mCustomerId;
      }

      /*public void setCustomerId(int mCustomerId) {
            this.mCustomerId = mCustomerId;
      }*/

      public void giveOrder(Item item, int numberOfPlates){
            //Order newOrder = new Order(mCustomerId);
            if (null == mOrder){
                  mOrder = new Order(mCustomerId);
            }
            ItemOrder newItemOrder = new ItemOrder(item,  numberOfPlates);
```

```
            mOrder.addOrder(newItemOrder);
      }


      public Order getOrder() {
            return mOrder;
      }

      public void setOrder(Order mOrder) {
            this.mOrder = mOrder;
      }

      public String getCustomerName() {
            return mCustomerName;
      }

      public void setCustomerName(String mCustomerName) {
            this.mCustomerName = mCustomerName;
      }

      public Customer findCustomer(int id ){

            if(id == mCustomerId){
                  return this;
            }
            return null;
      }

      public int getAllocatedTableId() {
            return allocatedTableId;
      }

      public void setAllocatedTableId(int allocatedTableId) {
            this.allocatedTableId = allocatedTableId;
      }
}
```

The Menu Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.ArrayList;
import java.util.Iterator;
```

```
import java.util.List;



public class Menu {
	private List<Item> mMenu;

	public Menu(){
		mMenu = new ArrayList<Item>();
		mMenu.add(new Item(11,"Tea", 5));
		mMenu.add(new Item(22, "Coffee", 10));
		mMenu.add(new Item(33, "Bread", 15));
	}
	public List<Item> getMenu(){
		return mMenu;
	}
	public void display(){
		int i = 0;
		Iterator<Item> it = mMenu.iterator();
		while(it.hasNext()){
			Item currentItem = it.next();
			System.out.println(i +":   " +currentItem.getItemId() + "    " +
currentItem.getItemName() + "    " + currentItem.getItemPrice());
			i++;
		}
	}

}
```

The Item Class

```
package com.somitsolutions.training.java.restaurant;

public class Item {
	private int mItemId;
	private String mName;
	private float mPrice;

	public Item(){

	}

	public Item(int id, String name, float price){
		mItemId = id;
```

```
                mName = name;
                mPrice = price;
        }
        public int getItemId() {
                return mItemId;
        }
        public void setItemId(int mItemId) {
                this.mItemId = mItemId;
        }
        public String getItemName() {
                return mName;
        }
        public void setItemName(String mName) {
                this.mName = mName;
        }
        public float getItemPrice() {
                return mPrice;
        }
        public void setItemPrice(float mPrice) {
                this.mPrice = mPrice;
        }
}
```

The ItemOrder Class

```
package com.somitsolutions.training.java.restaurant;

public class ItemOrder {
        private Item mItem;
        private int mNumberOfPlates;


        public ItemOrder(){
                mItem = null;
                mNumberOfPlates = 0;
        }
        public ItemOrder(Item item, int numberOfPlates){
                mItem = item;
                mNumberOfPlates = numberOfPlates;
        }

        public Item getItem() {
                return mItem;
        }
        public void setItem(Item mItem) {
                this.mItem = mItem;
```

```
	}
	public int getNumberOfPlates() {
		return mNumberOfPlates;
	}
	public void setNumberOfPlates(int mNumberOfPlates) {
		this.mNumberOfPlates = mNumberOfPlates;
	}
}
```

The Order Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.ArrayList;
import java.util.List;

public class Order {
	private int mOrderId;//this is the same as customerId imagining
					//that one customer will create only one order
	private List<ItemOrder> mItemOrder = new ArrayList<ItemOrder>();

	public Order(){
		mOrderId = 0;
		//mItemOrder = null;
	}
	public Order(int orderId){
		mOrderId = orderId;
	}

	public List<ItemOrder> getItemOrder() {
		return mItemOrder;
	}

	public void setItemOrder(List<ItemOrder> mItemOrder) {
		this.mItemOrder = mItemOrder;
	}

	public int getOrderId() {
		return mOrderId;
	}

	public void setOrderId(int mOrderId) {
		this.mOrderId = mOrderId;
	}

	public void addOrder(ItemOrder itemOrder){
```

```
            mItemOrder.add(itemOrder);
      }
}
```

The Bill Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.Iterator;


public class Bill {
      //private Order mOrder;
      private int mBillId;//this is the same as orderId
      private Customer mCustomer;

      public Bill(int Id, Customer customer){
            mBillId = Id;
            mCustomer = customer;

      }

      public float calculateTotal(){

            float retValue = 0;
            Iterator<ItemOrder> it = mCustomer.getOrder().getItemOrder().iterator();

            while (it.hasNext() == true){
                  ItemOrder element = it.next();
                  retValue+= (element.getItem().getItemPrice())*
(element.getNumberOfPlates());
            }
            return retValue;
      }
      public int getBillId() {
            return mBillId;
      }
      public void setBillId(int mBillId) {
            this.mBillId = mBillId;
      }

}
```

The Table Class

```
package com.somitsolutions.training.java.restaurant;
```

```
import java.util.Observable;

public class Table extends Observable{

	private boolean mOccupied;

	public Table(){
		mOccupied = false;
	}

	public boolean isOccupied(){
		return
				mOccupied == true;
	}

	public void bookTable(){
		mOccupied = true;

	}

	public void releaseTable(){
		mOccupied = false;
		setChanged();
		notifyObservers();

	}

}
```

The Receptionist Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;
import java.util.Scanner;

public class Receptionist extends Employee implements Observer{

	Scanner myScan = new Scanner(System.in);
	private Customer currentCustomer;
	private Customer lastCustomerServed;

	public Receptionist(){
```

```
            currentCustomer = null;
      }

      public Receptionist(int empId, String name){
            super(empId, name);
            currentCustomer = null;
      }

      public boolean isAnyTableOccupied(){
            for(int i = 0; i < Restaurant.getRestaurant().MAX_NUMBER_OF_TABLES; i++){
                  if (Restaurant.getRestaurant().getTables().get(i).isOccupied() ==
true){
                        return true;
                  }
            }
            return false;
      }

      public int findTheEmptyTable(){

            for(int i = 0; i < Restaurant.getRestaurant().MAX_NUMBER_OF_TABLES; i++){
                  if(Restaurant.getRestaurant().getTables().get(i).isOccupied() ==
false){
                        return i;
                  }
            }
            return -1;
      }

      public void displayOccupiedTable(){
            for (int i = 0; i <Restaurant.getRestaurant().MAX_NUMBER_OF_TABLES; i++){
                  if(Restaurant.getRestaurant().getTables().get(i).isOccupied() ==
true){
                        System.out.println("Table No. " + i + " is occupied");
                  }
            }
      }

      public boolean isAllTableOccupied(){
            int val = findTheEmptyTable();
            if (val == -1){
                  return true;
            }
            else {
                  return false;
            }
      }
```

```
	public void bookATable(){
		if(isAllTableOccupied() == false){
			int tableNumber = findTheEmptyTable();

			//get the customer from the list in the FIFO order
			//and remove him from the queue
			if(!Restaurant.getRestaurant().getCustomerQueue().isEmpty()){


				currentCustomer =
				Restaurant.getRestaurant().getCustomerQueue().pollLast();

				currentCustomer.setAllocatedTableId(tableNumber);

				Restaurant.getRestaurant().addCustomerToTheServingArray(current
				Customer);
				Restaurant.getRestaurant().bookTable(tableNumber);
				System.out.println(currentCustomer.getCustomerName() + " has
been given table number: "+ tableNumber);
			}
		}

	}

	public void addNewCustomer(Customer customer){

			Restaurant.getRestaurant().getCustomerQueue().offerFirst(customer);
		}

	@Override
	public void update(Observable o, Object arg) {
		// TODO Auto-generated method stub

//System.out.println(Restaurant.getRestaurant().getCustomerQueue().pollLast().getCustomer
Name());
		if(!Restaurant.getRestaurant().getCustomerQueue().isEmpty()){
			bookATable();
		}

	}

	public Customer getCurrentCustomer() {
		return currentCustomer;
	}

	public void setCurrentCustomer(Customer currentCustomer) {
		this.currentCustomer = currentCustomer;
```

```
	}

	public Customer createNewCustomer(){
		Customer customer = new Customer();
		//System.out.println(customer.getCustomerId());
		System.out.println("Please enter customer name...");
		String customerName = myScan.nextLine();
		customer.setCustomerName(customerName);

		addNewCustomer(customer);

		return customer;
	}

	public void sayGoodBye(Customer customer){
		Restaurant.getRestaurant().getCustomerArray().remove(customer);
		Restaurant.getRestaurant().getCustomerServed().add(customer);
	}

	public Customer getLastCustomerServed() {
		return lastCustomerServed;
	}

	public void setLastCustomerServed(Customer lastCustomerServed) {
		this.lastCustomerServed = lastCustomerServed;
	}
}
```

The Waiter Class

```
package com.somitsolutions.training.java.restaurant;

import java.util.Iterator;

public class Waiter extends Employee {

	public Waiter(){

	}

	public Waiter(int empId, String name){

		super(empId, name);
	}


	public void displayMenu(){
```

```
            Restaurant.getRestaurant().getMenu().display();
      }

      public void releaseTable(int tableId){
            if(Restaurant.getRestaurant().getTables().get(tableId).isOccupied() ==
true){
                  Iterator it =
Restaurant.getRestaurant().getCustomerArray().iterator();
                  while(it.hasNext()){
                        Customer c = (Customer) it.next();
                        if(tableId == c.getAllocatedTableId()){

Restaurant.getRestaurant().getReceptionist().setLastCustomerServed(c);
                        }
                  }
                  Restaurant.getRestaurant().releaseTable(tableId);
            }


            if ((Restaurant.getRestaurant().getCustomerQueue().isEmpty() == false) ||
(Restaurant.getRestaurant().getReceptionist().isAnyTableOccupied() == true)){
                  System.out.println("There are still some customers...");
                  /**/
            }

            if ((Restaurant.getRestaurant().getCustomerQueue().isEmpty() == true) &&
(Restaurant.getRestaurant().getReceptionist().isAnyTableOccupied() == false)){
                  System.out.println("All served...");
                  Restaurant.getRestaurant().getCustomerArray().clear();
            }

      }

}
```

The Casher Class

```
package com.somitsolutions.training.java.restaurant;

public class Cashier extends Employee{

      public Cashier(){

      }

      public Cashier(int empId, String name){
```

```
            super(empId, name);
      }

      public void generateBill(Customer customer){
            //Customer customer = Restaurant.getRestaurant().getCustomer(customer);
            //System.out.println(customer.getCustomerName());
            Bill newBill = new Bill(customer.getCustomerId(),customer);
            Restaurant.getRestaurant().addNewBill(newBill);
            System.out.println("Total Amount for Customer :" +
customer.getCustomerName() + " is " + newBill.calculateTotal());
      }

      public void sayGoodBye(Customer customer){
            //Customer customerToBeRemoved =
Restaurant.getRestaurant().getCustomer(customerId);
            Restaurant.getRestaurant().getReceptionist().sayGoodBye(customer);
      }
}
```

The Main RestaurantSystem Class

```
package com.somitsolutions.training.java.restaurant;


import java.util.Scanner;

public class RestaurantSystem {
      public static void main(String[] args) {

            Scanner myScan = new Scanner(System.in);
            String s = "Y";
            Restaurant restaurant = Restaurant.getRestaurant();
            do{
                  int tableId = 0;
                  //int customerId = 0;
                  Customer customer = null;
                        System.out.println("Is there a new customer?");
                        String ans = myScan.nextLine();
                        if(ans.equalsIgnoreCase("Y")){
                              customer =
restaurant.getReceptionist().createNewCustomer();
                              System.out.println(customer.getCustomerId());
                              if(restaurant.getReceptionist().isAllTableOccupied() ==
false){
                                    restaurant.getReceptionist().bookATable();
                                    restaurant.getWaiter().displayMenu();
```

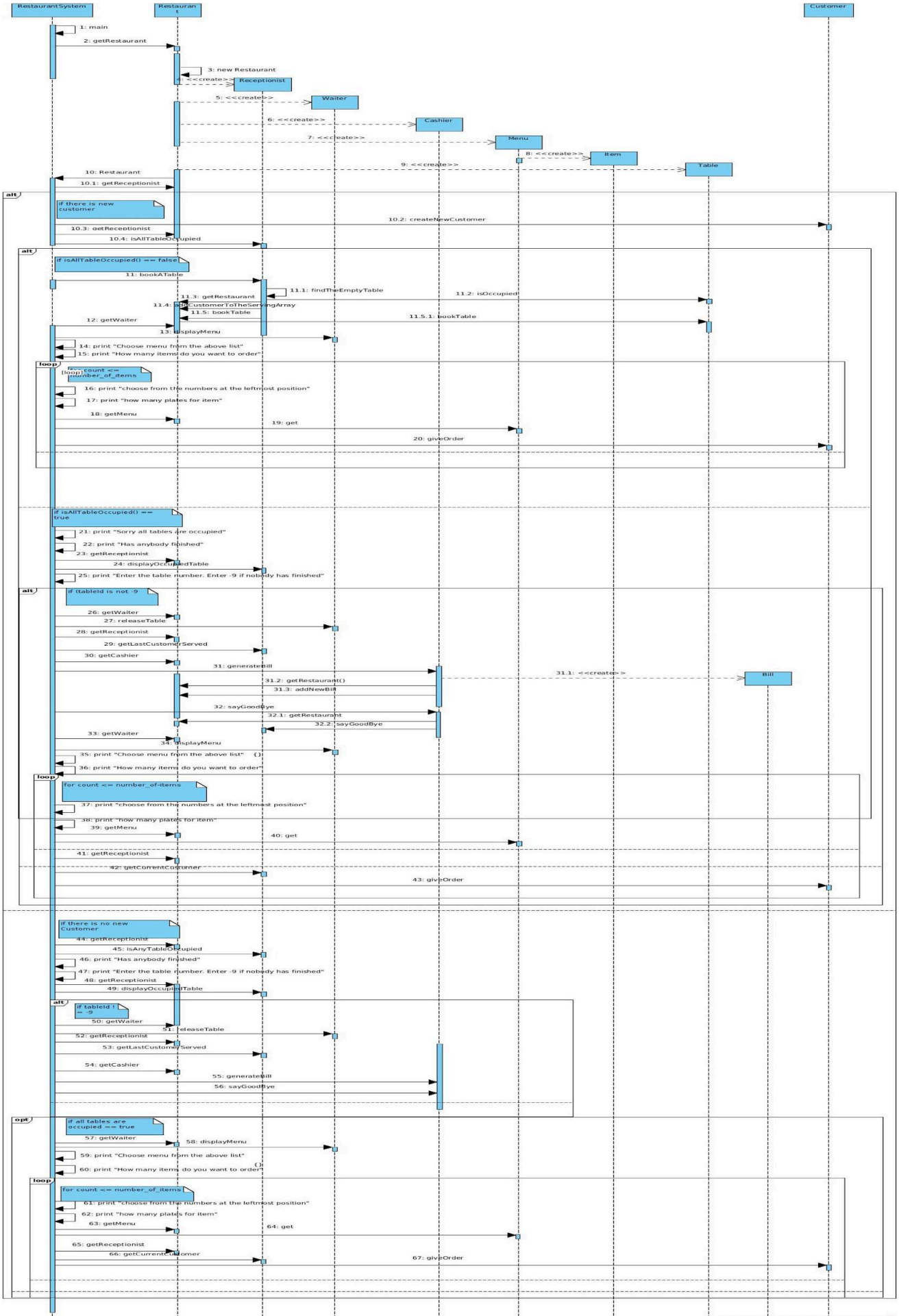
```
                                        System.out.println("Choose Menu from the above
List");
                                        System.out.println("How many items do you want to
order?");
                                        int number_of_items = myScan.nextInt();
                                        for (int i = 0; i<number_of_items; i++){
                                               System.out.println("Choose from the numbers
at the leftmost position");
                                               int item_position = myScan.nextInt();
                                               System.out.println("How many plates of menu
item " + item_position + " you want to order");
                                               int number_of_plates = myScan.nextInt();
                                               if(i == number_of_items - 1){
                                                      myScan.nextLine();
                                               }

                                               Item item =
restaurant.getMenu().getMenu().get(item_position);
                                               customer.giveOrder(item, number_of_plates);
                                        }
                                 }
                                 else {
                                        System.out.println("Sorry all tables are
occupied... Please wait...");
                                        System.out.println("Has anybody finished?");
                                        System.out.println("Occupied tables are:");

restaurant.getReceptionist().displayOccupiedTable();
                                        System.out.println("Enter the Table number. Enter
-9 if no one has finished...");
                                        tableId = Integer.parseInt(myScan.nextLine());

                                        if(tableId != -9 ){


                                               //System.out.println()

restaurant.getWaiter().releaseTable(tableId);
                                               Customer customerJustServed =
Restaurant.getRestaurant().getReceptionist().getLastCustomerServed();

System.out.println(customerJustServed.getCustomerName());

System.out.println(customerJustServed.getCustomerId());
```

```
restaurant.getCashier().generateBill(customerJustServed);

restaurant.getCashier().sayGoodBye(customerJustServed);


                                        restaurant.getWaiter().displayMenu();
                                        System.out.println("Choose Menu from the
above List");
                                        System.out.println("How many items do you
want to order?");
                                        int number_of_items = myScan.nextInt();
                                        for (int i = 0; i<number_of_items; i++){
                                              System.out.println("Choose from the
numbers at the leftmost position");
                                              int item_position =
myScan.nextInt();
                                              System.out.println("How many plates
of menu item " + item_position + " you want to order");
                                              int number_of_plates =
myScan.nextInt();
                                              if(i == number_of_items - 1){
                                                    myScan.nextLine();
                                              }

                                              Item item =
restaurant.getMenu().getMenu().get(item_position);
                                              Customer current =
restaurant.getReceptionist().getCurrentCustomer();

System.out.println(current.getCustomerName());

//System.out.println(current.getCustomerName());
                                              current.giveOrder(item,
number_of_plates);
                                        }
                                  }

                                  else{
                                        continue;
                                  }
                            }
                      }
                      else {
```

```
                                if(restaurant.getReceptionist().isAnyTableOccupied() ==
true){
                                //System.out.println("There are still some customers");
                                System.out.println("Has anybody finished?");
                                System.out.println("Enter the table number. Enter -9 if
no one has finished...");
                                System.out.println("Occupied tables are:");
                                restaurant.getReceptionist().displayOccupiedTable();
                                int numberTable = myScan.nextInt();
                                myScan.nextLine();
                                if(numberTable != -9){

                                        restaurant.getWaiter().releaseTable(numberTable);
                                        Customer customerJustServed =
Restaurant.getRestaurant().getReceptionist().getLastCustomerServed();

restaurant.getCashier().generateBill(customerJustServed);

restaurant.getCashier().sayGoodBye(customerJustServed);


if(restaurant.getReceptionist().isAllTableOccupied() == false){
                                                continue;
                                        }
                                        restaurant.getWaiter().displayMenu();
                                        System.out.println("Choose Menu from the above
List");
                                        System.out.println("How many items do you want to
order?");
                                        int number_of_items = myScan.nextInt();
                                        for (int i = 0; i<number_of_items; i++){
                                                System.out.println("Choose from the numbers
at the leftmost position");
                                                int item_position = myScan.nextInt();
                                                System.out.println("How many plates of menu
item " + item_position + " you want to order");
                                                int number_of_plates = myScan.nextInt();
                                                if(i == number_of_items - 1){
                                                        myScan.nextLine();
                                                }

                                                Item item =
restaurant.getMenu().getMenu().get(item_position);
                                                Customer current =
restaurant.getReceptionist().getCurrentCustomer();
```

```
//System.out.println(current.getCustomerName());
                                                current.giveOrder(item, number_of_plates);
                                        }


                                        ////////////////
                                        }
                                else {
                                        continue;
                                }

                                }
                        }
                System.out.println("Do you want to continue?");
                s = myScan.nextLine();

                if(s.equalsIgnoreCase("N")){
                        System.exit(0);
                }

        }while (s.equalsIgnoreCase("Y"));
    }
}
```

The Sequence Diagram


RestaurantSystem
Restauran t
Customer
1: main
2: getRestaurant
3: new Restaurant
<<create>>
Receptionist
5: <<create>>
Waiter
6: <<create>>
Cashier
7: <<create>>
Menu
8: <<create>>
Item
9: <<create>>
Table
10: Restaurant
10.1: getReceptionist
alt
if there is new customer
10.2: createNewCustomer
10.3: getReceptionist
10.4: isAllTableOccupied
alt
if isAllTableOccupied() == false
11: bookATable
11.1: findTheEmptyTable
11.2: isOccupied
11.3: getRestaurant
11.4: addCustomerToTheServingArray
11.5: bookTable
11.5.1: bookTable
12: getWaiter
13: displayMenu
14: print "Choose menu from the above list"
15: print "How many items do you want to order"
loop
[loop] count <= number_of_items
16: print "choose from the numbers at the leftmost position"
17: print "how many plates for item"
18: getMenu
19: get
20: giveOrder
if isAllTableOccupied() == true
21: print "Sorry all tables are occupied"
22: print "Has anybody finished"
23: getReceptionist
24: displayOccupiedTable
25: print "Enter the table number. Enter -9 if nobody has finished"
alt
if (tableId is not -9
26: getWaiter
27: releaseTable
28: getReceptionist
29: getLastCustomerServed
30: getCashier
31: generateBill
31.1: <<create>>
Bill
31.2: getRestaurant()
31.3: addNewBill
32: sayGoodBye
32.1: getRestaurant
32.2: sayGoodBye
33: getWaiter
34: displayMenu
35: print "Choose menu from the above list" {}
36: print "How many items do you want to order"
loop
for count <= number_of_items
37: print "choose from the numbers at the leftmost position"
38: print "how many plates for item"
39: getMenu
40: get
41: getReceptionist
42: getCurrentCustomer
43: giveOrder
if there is no new Customer
44: getReceptionist
45: isAnyTableOccupied
46: print "Has anybody finished"
47: print "Enter the table number. Enter -9 if nobody has finished"
48: getReceptionist
49: displayOccupiedTable
alt
if tableId != -9
50: getWaiter
51: releaseTable
52: getReceptionist
53: getLastCustomerServed
54: getCashier
55: generateBill
56: sayGoodBye
opt
if all tables are occupied == true
57: getWaiter
58: displayMenu
59: print "Choose menu from the above list"
60: print "How many items do you want to order" {}
loop
for count <= number_of_items
61: print "choose from the numbers at the leftmost position"
62: print "how many plates for item"
63: getMenu
64: get
65: getReceptionist
66: getCurrentCustomer
67: giveOrder