CENG 4513 Modeling and Simulation
2021-2022 Fall Assignment 02
Case Study with Java Agent Development Env.(JADE)

Self service Coffee Shop Modeling Report

Group Members: Mert Furkan Ergüden
Mehmet Kubilay Elüstü
Zeynep Hazal Cengiz

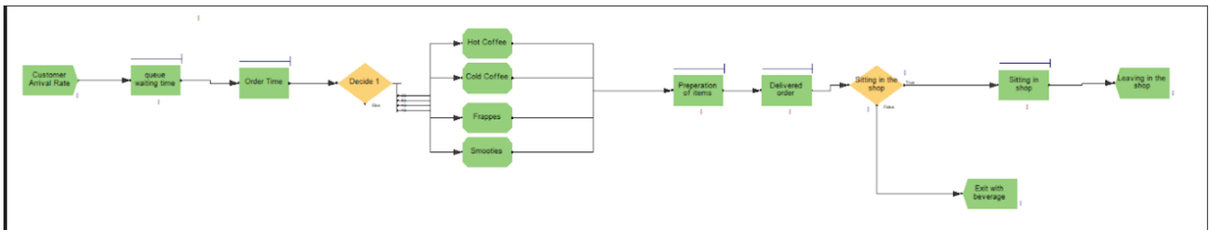
Introduction

In this project, we demonstrated the real life problem on simulation using AREANA software for the Modeling and Simulation course with the course code Ceng 4513. In this project, we realized it using a real scenario and using the JADE library.
Our real time project topic is Self Service Coffee Shop system. Self-service cafes are a very common coffeehouse culture around the world. With this culture, people can order in coffee shops without the need for employees such as waiters. We had done the necessary work on this topic that we chose in Assignment 1. However, in this project, the main thing is to make our project as a simulation with JADE.We used the Java programming language throughout this project. You can find the source codes in the codes folder in the relevant zip file. We identified the relevant assets in our project with the JADE library. We clustered these identified entities in modules and worked on our simulation of certain decisions.
We used the JADE library throughout the project. JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases.

Report Operation

While we were writing the report, we did not write it by taking into account the task numbers of the given tasks. However, it was aimed to do all the tasks in an unordered manner and write them in the report.

We tried to design a FIPA-compliant agent-based simulation system.
We used triangular distribution for transitions between models. To make the simulation we prepared with Arena a structure that can be used in real life with the JAVA programming language.

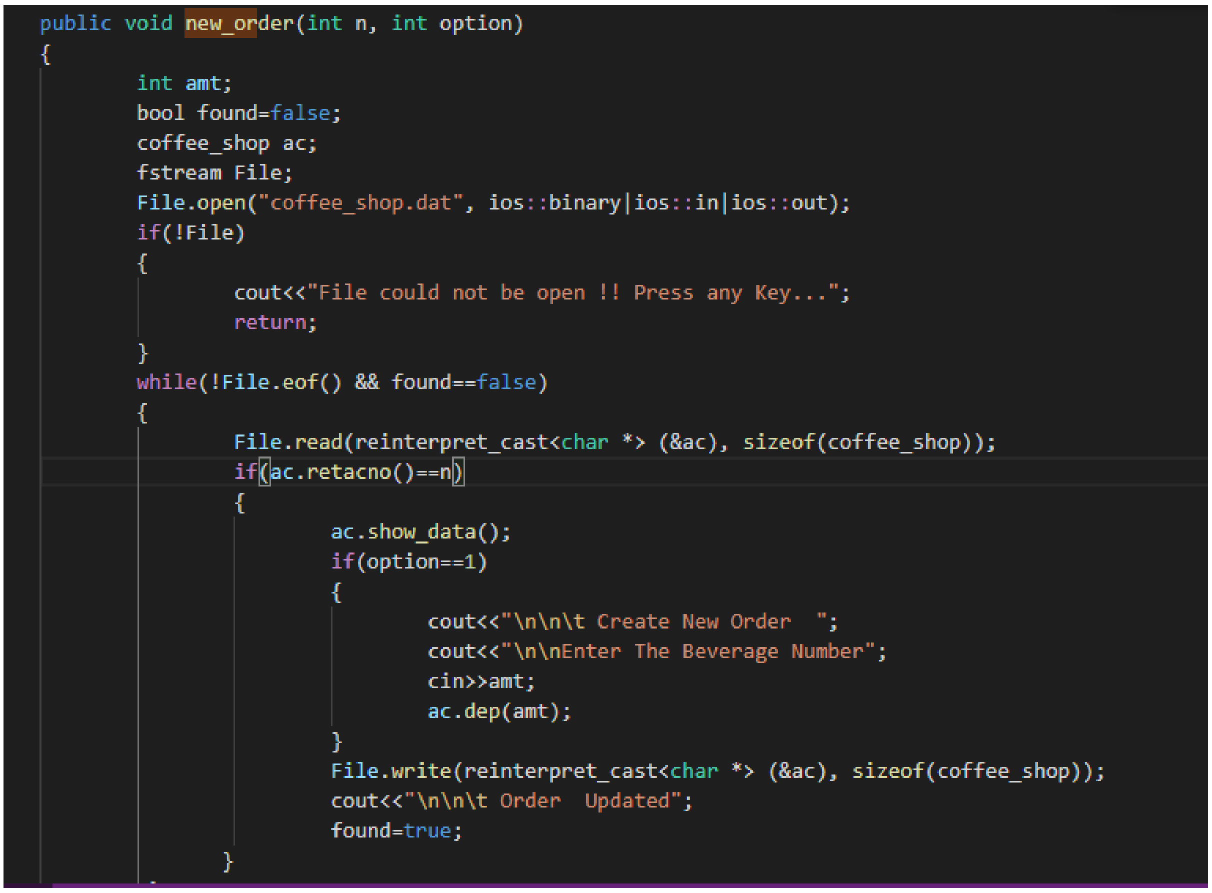


General screenshot link

What you see above is the general screenshot of our model. We did this task in our previous project. However, while coding, we did our coding by considering the general screen image. Thus, the coding process was easy, as we were constantly examining the system while coding.

While coding, we implemented all of the java codes in a single file and class. We named the class we use as coffee_shop. We called the relevant methods in this class in order. Because we aimed to work in a proper structure.

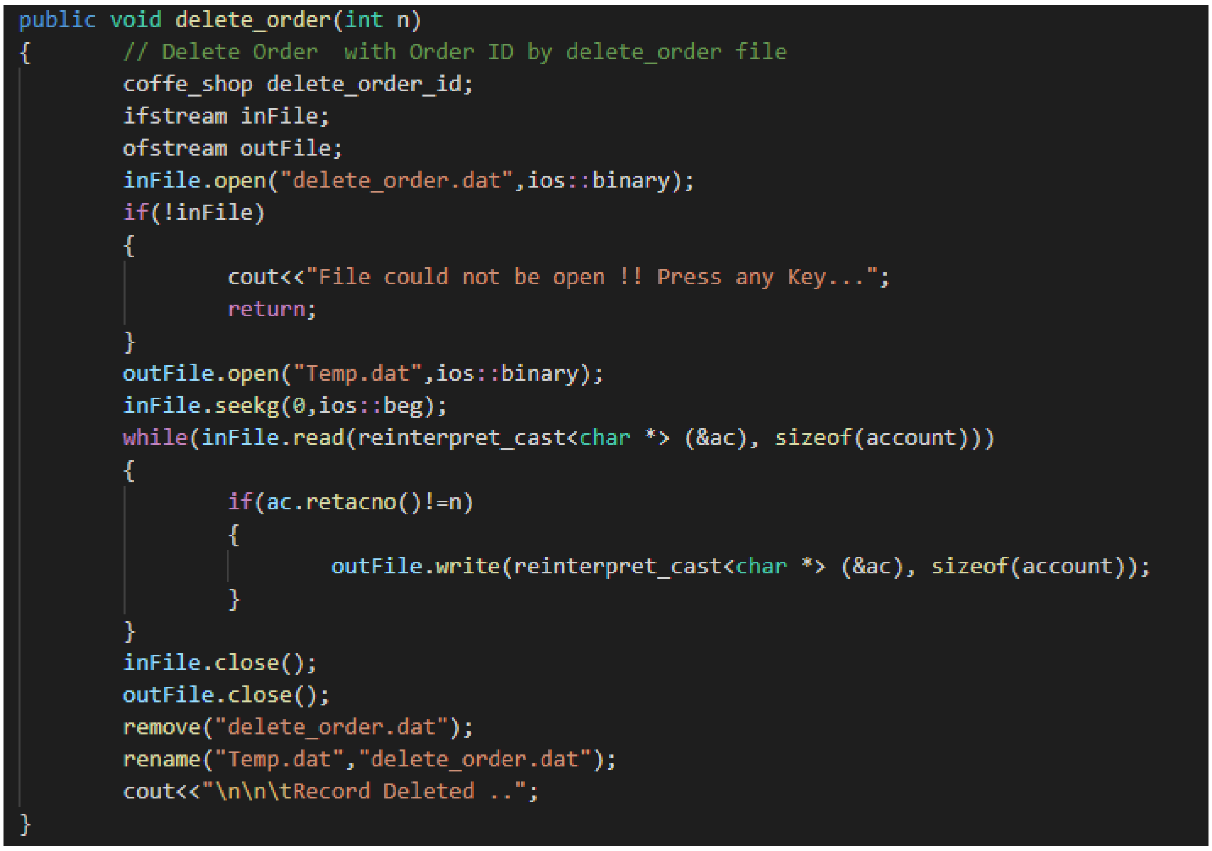
We created methods for each of our activities. For example, when ordering on the menu, we run the create_order method. This method allows us to create a new order. We can add the created orders into the orders.dat file (you can also use it in .txt file format if you want). We make the orders with the number of the relevant drink on the menu.



```
public void new_order(int n, int option)
{
        int amt;
        bool found=false;
        coffee_shop ac;
        fstream File;
        File.open("coffee_shop.dat", ios::binary|ios::in|ios::out);
        if(!File)
        {
                cout<<"File could not be open !! Press any Key...";
                return;
        }
        while(!File.eof() && found==false)
        {
                File.read(reinterpret_cast<char *> (&ac), sizeof(coffee_shop));
                if(ac.retacno()==n)
                {
                        ac.show_data();
                        if(option==1)
                        {
                                cout<<"\n\n\t Create New Order  ";
                                cout<<"\n\nEnter The Beverage Number";
                                cin>>amt;
                                ac.dep(amt);
                        }
                        File.write(reinterpret_cast<char *> (&ac), sizeof(coffee_shop));
                        cout<<"\n\n\t Order  Updated";
                        found=true;
               }
```

create_order method

We do the same with deleting an order. There is a slight difference in the order deletion process. We store the deleted orders in the delete_order.dat file.

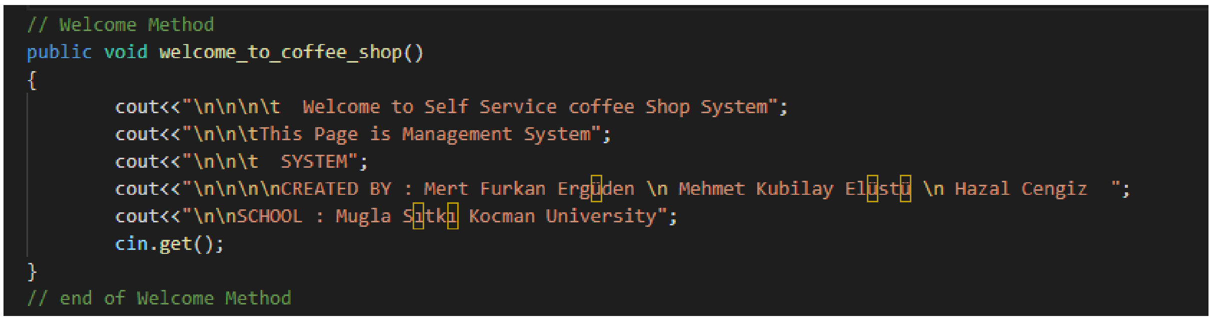


```
public void delete_order(int n)
{       // Delete Order  with Order ID by delete_order file
        coffe_shop delete_order_id;
        ifstream inFile;
        ofstream outFile;
        inFile.open("delete_order.dat",ios::binary);
        if(!inFile)
        {
                cout<<"File could not be open !! Press any Key...";
                return;
        }
        outFile.open("Temp.dat",ios::binary);
        inFile.seekg(0,ios::beg);
        while(inFile.read(reinterpret_cast<char *> (&ac), sizeof(account)))
        {
                if(ac.retacno()!=n)
                {
                        outFile.write(reinterpret_cast<char *> (&ac), sizeof(account));
                }
        }
        inFile.close();
        outFile.close();
        remove("delete_order.dat");
        rename("Temp.dat","delete_order.dat");
        cout<<"\n\n\tRecord Deleted ..";
}
```

delete_order method

The processes we mentioned were about adding the orders selected from the basic menu as an order according to the beverage numbers or managing the adding orders.We have added at least two models to be prepared to fill in.

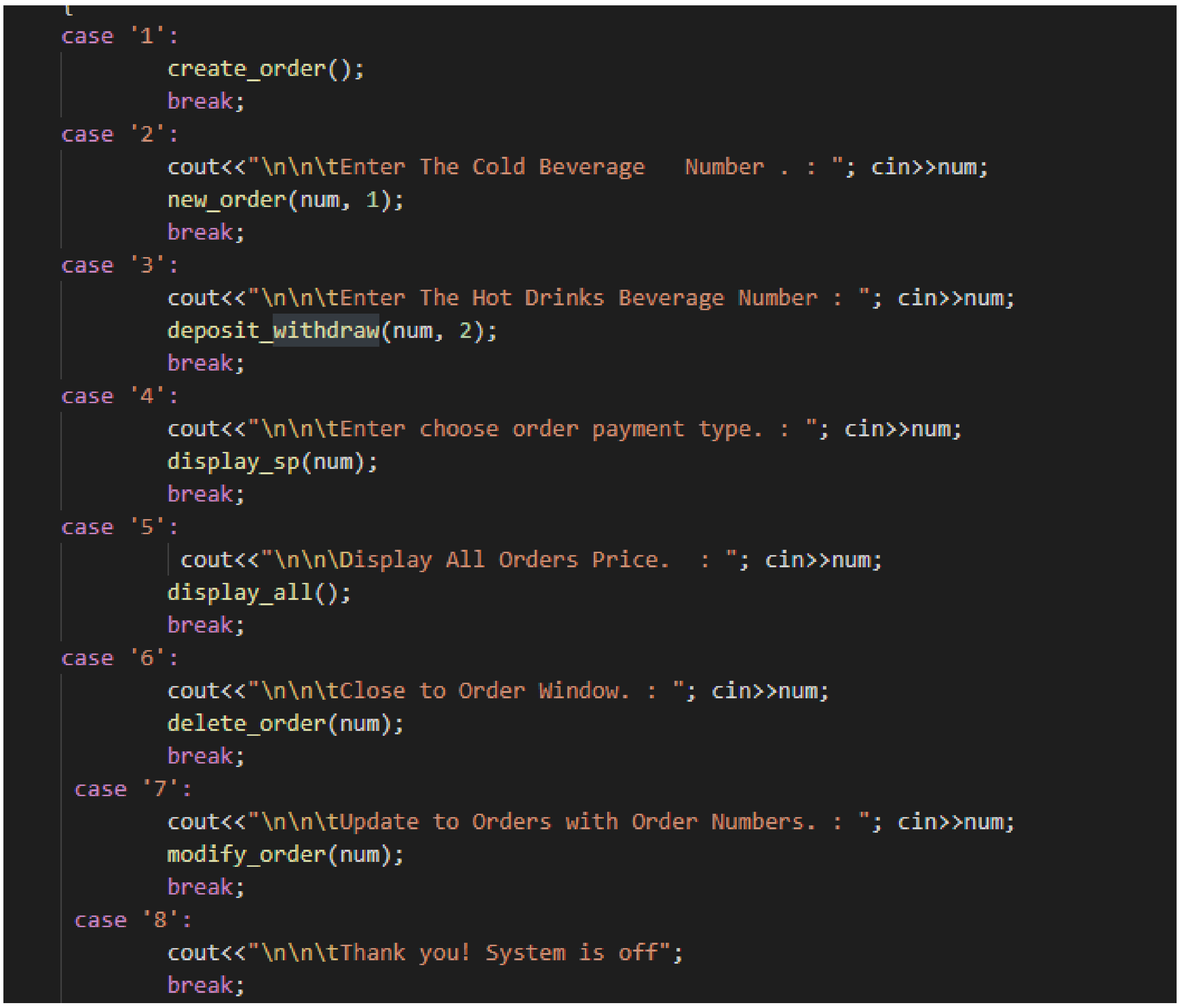
We mentioned earlier that more than one method works on the model we have prepared. When our model first runs, the welcome method welcomes us. With this method, we aimed to give a little information to the people who use the system. Thus, we aimed to make the system we modeled more useful.



```
// Welcome Method
public void welcome_to_coffee_shop()
{
        cout<<"\n\n\n\t  Welcome to Self Service coffee Shop System";
        cout<<"\n\n\tThis Page is Management System";
        cout<<"\n\n\t  SYSTEM";
        cout<<"\n\n\n\nCREATED BY : Mert Furkan Ergüden \n Mehmet Kubilay Elüstü \n Hazal Cengiz  ";
        cout<<"\n\nSCHOOL : Mugla Sıtkı Kocman University";
        cin.get();
}
// end of Welcome Method
```

welcome method

When the system starts up for the first time, little information is given on the user interface and then the system listens to the actions that the user wants to do on the system. We did this with the following codes:

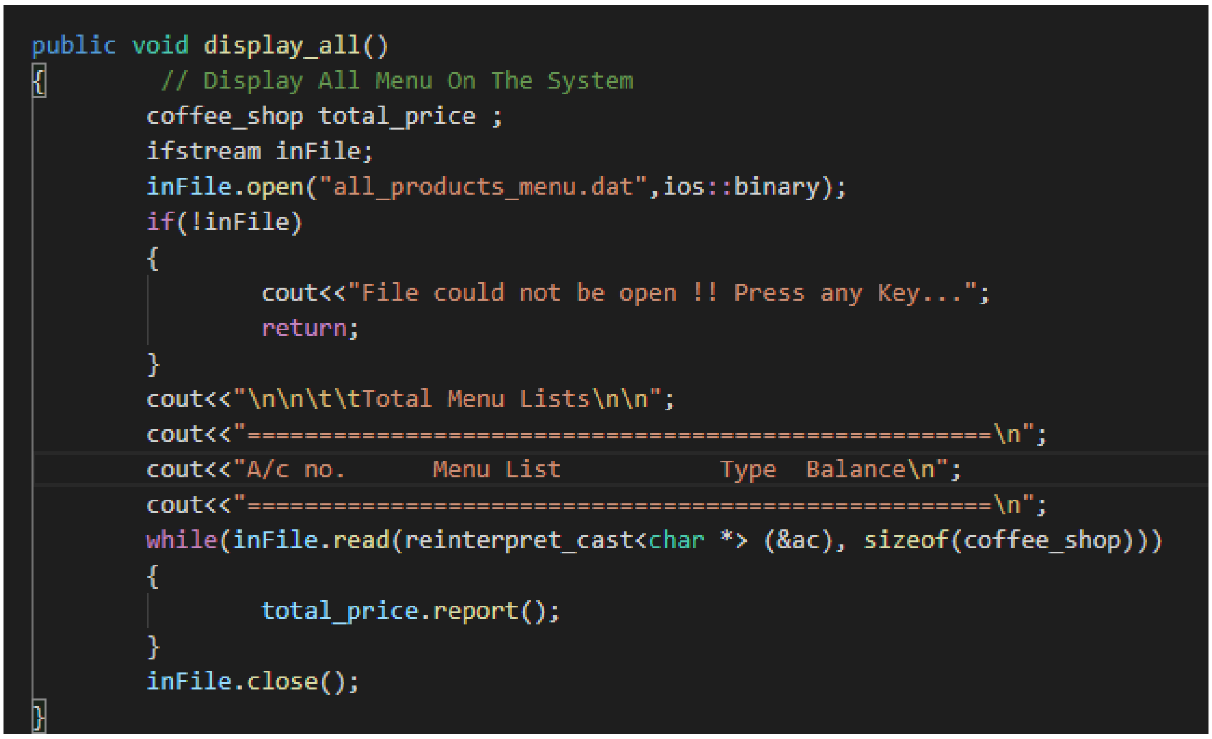


```
case '1':
        create_order();
        break;
case '2':
        cout<<"\n\n\tEnter The Cold Beverage   Number . : "; cin>>num;
        new_order(num, 1);
        break;
case '3':
        cout<<"\n\n\tEnter The Hot Drinks Beverage Number : "; cin>>num;
        deposit_withdraw(num, 2);
        break;
case '4':
        cout<<"\n\n\tEnter choose order payment type. : "; cin>>num;
        display_sp(num);
        break;
case '5':
         cout<<"\n\n\Display All Orders Price.  : "; cin>>num;
        display_all();
        break;
case '6':
        cout<<"\n\n\tClose to Order Window. : "; cin>>num;
        delete_order(num);
        break;
 case '7':
        cout<<"\n\n\tUpdate to Orders with Order Numbers. : "; cin>>num;
        modify_order(num);
        break;
 case '8':
        cout<<"\n\n\tThank you! System is off";
        break;
```

We used the switch case method in this area. With this method, when the system is running, different methods are called upon requests from users 1 to 8, which come to the user interface, and the system works.
If the User presses the following numbers in order;

- 1 => a new order is created.
- 2 => Cold drinks section is shown from the menu
- 3 => Hot drinks section is displayed from the menu.
- 4 => Order payment processing
- 5 => Show order total amount
- 6 => Close orders file
- 7 => Edit orders
- 8 => Exit system

If the user wants to access all the menus in the system, he can access the desired information with the file all_products_menu.dat.



```
public void display_all()
{        // Display All Menu On The System
        coffee_shop total_price ;
        ifstream inFile;
        inFile.open("all_products_menu.dat",ios::binary);
        if(!inFile)
        {
                cout<<"File could not be open !! Press any Key...";
                return;
        }
        cout<<"\n\n\t\tTotal Menu Lists\n\n";
        cout<<"====================================================\n";
        cout<<"A/c no.      Menu List           Type  Balance\n";
        cout<<"====================================================\n";
        while(inFile.read(reinterpret_cast<char *> (&ac), sizeof(coffee_shop)))
        {
                total_price.report();
        }
        inFile.close();
}
```

display all menu products

In this part, we kept all the menu contents in a file that we determined. By calling this file whenever we want, we can see the total amount of products in the system, that is, all of the menus.Thus, we are able to perform multiple operations on the system.

While we were converting our simulation to code, we took into account the needs of users using the system. These needs aim to provide a more beneficial service to customers. In line with these goals, the satisfaction of the customers can also be considered as the next project.

Conclusion

This project aimed at new methodologies not only for designing, analyzing and optimizing self-service coffee shop systems, but also easily applicable to various other service industries where customer behavior is of fundamental importance. We took our processes to the next level and coded them with the JADE library. We used the Java programming language throughout the project. With this project, we took our knowledge of Java to a higher level. We aimed for cafe staff to use our system. In line with the requested orders, we have made many transactions in the system. We wanted to seek answers to these questions by making a system simulation. Simulated systems allow the system to be understood and analyzed as a whole, not as any part of it. The results of the simulation provide insight into the various possible outcomes and how to manage them. By coding this system we have created with the programming language (JAVA) we have determined, we have taken the system a little further.