Pizza Ordering System

Team 7

Ahmed Maghrbi

Azadeh Ayani

Aaron Leff

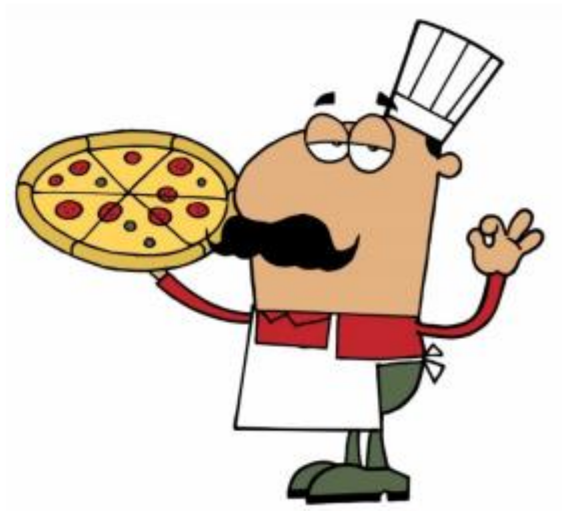


Table of Contents

Introduction

Topic Description and Requirements

We would like to create a pizza ordering system that allows the cashier to enter customer order into the system. The cashier can create, display, update, and delete orders as needed. The customer will be order to choose a several different kinds of pizzas, choosing different toppings and/or bread types. The customer can select one of three sizes: small, medium and large. The price varies according to the size and type of pizza. The cashier will accept different types of payment from the customer (credit, cash and coupon). The cashier will generate sales reports daily and monthly. The application will be programmed using Java.

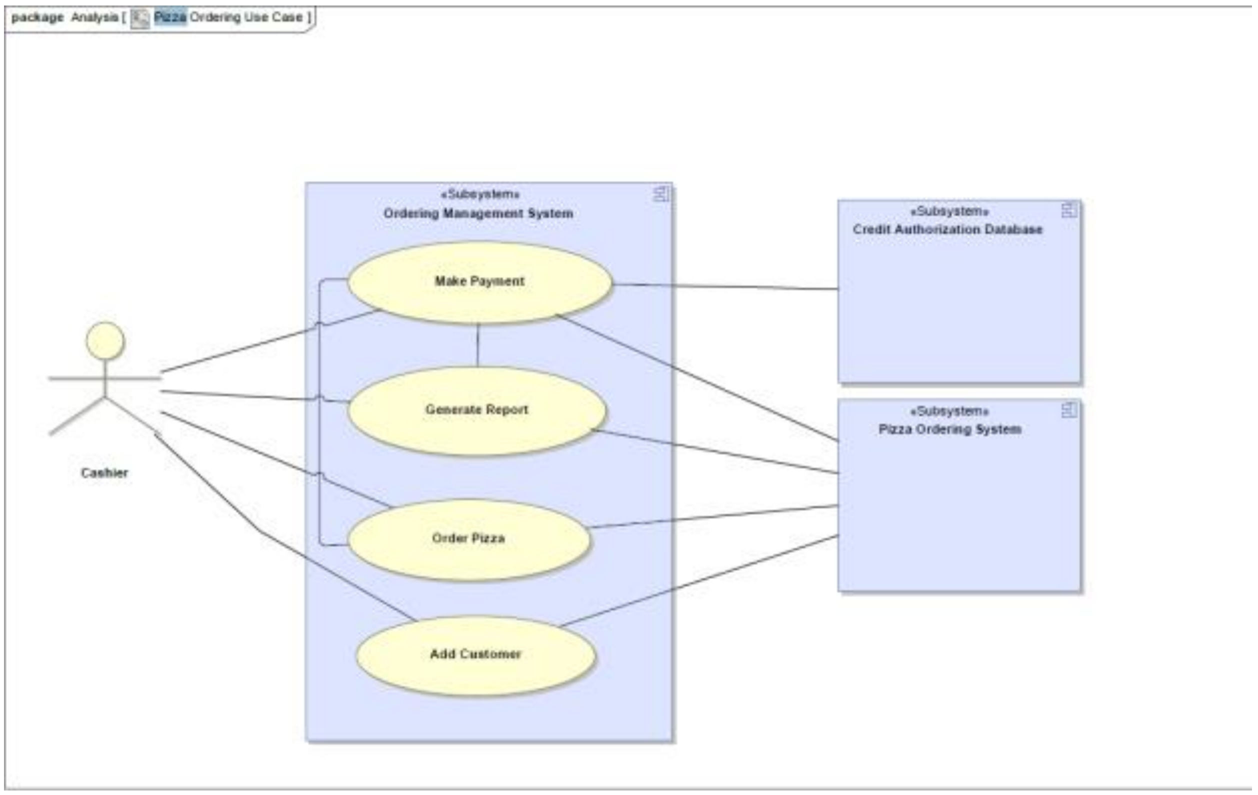
We established the following main requirements for our project:

- Creating basic pizza ordering system
- Cashier has ability to enter orders as needed
- System should be able to do both delivery and pick-up
- There are 3 sizes for our pizzas
- Customer can order multiple pizzas
- Customer can customize their pizza
- There are three payment options: Cash, Credit Card, and Coupon
- System stores customer’s information
- Cashier can produce daily/monthly reports

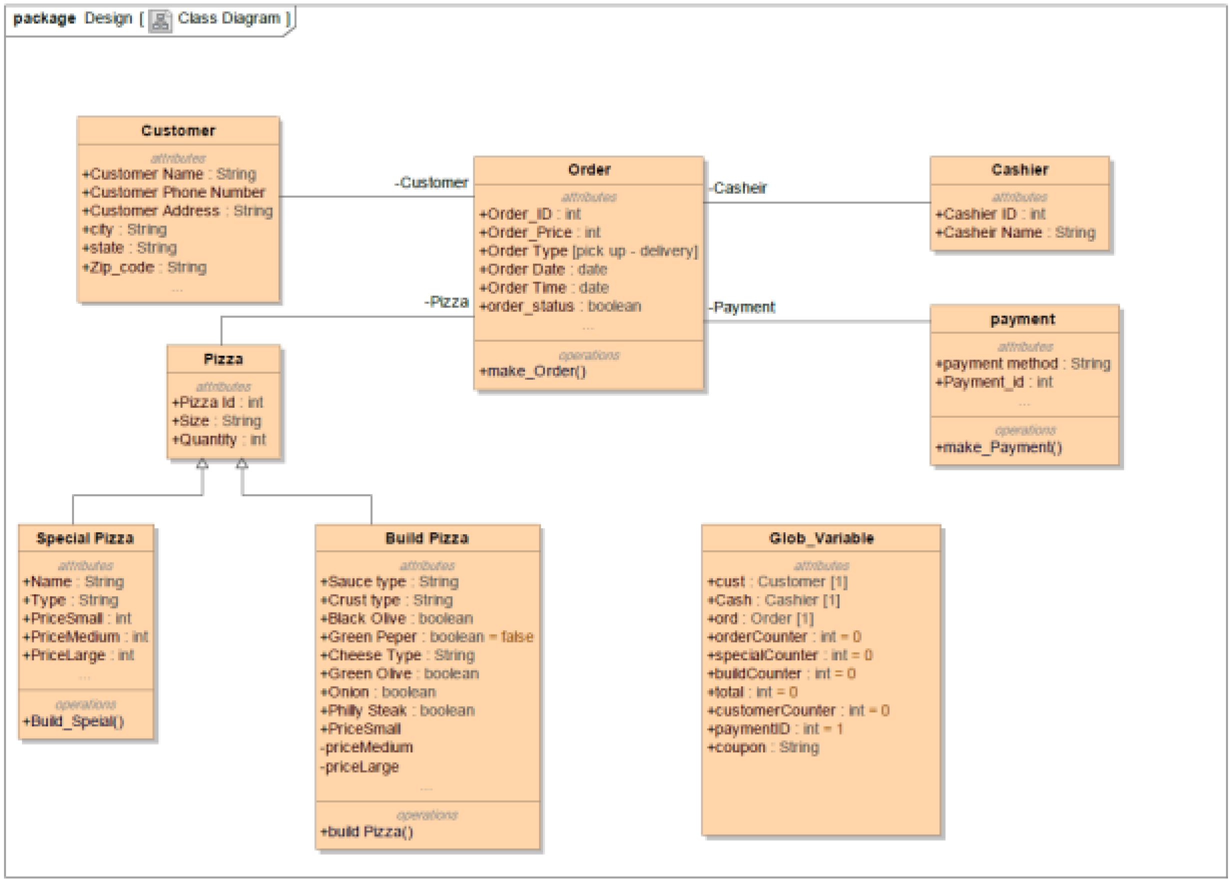
Analysis Artifacts

The Analysis Artifacts include the following: 1) the Use Case diagram; 2) the Domain Model; 3) three System Sequence Diagrams (Order, Payment, and Report); and 4) Fully-Dressed documents for each Use Case SSD.

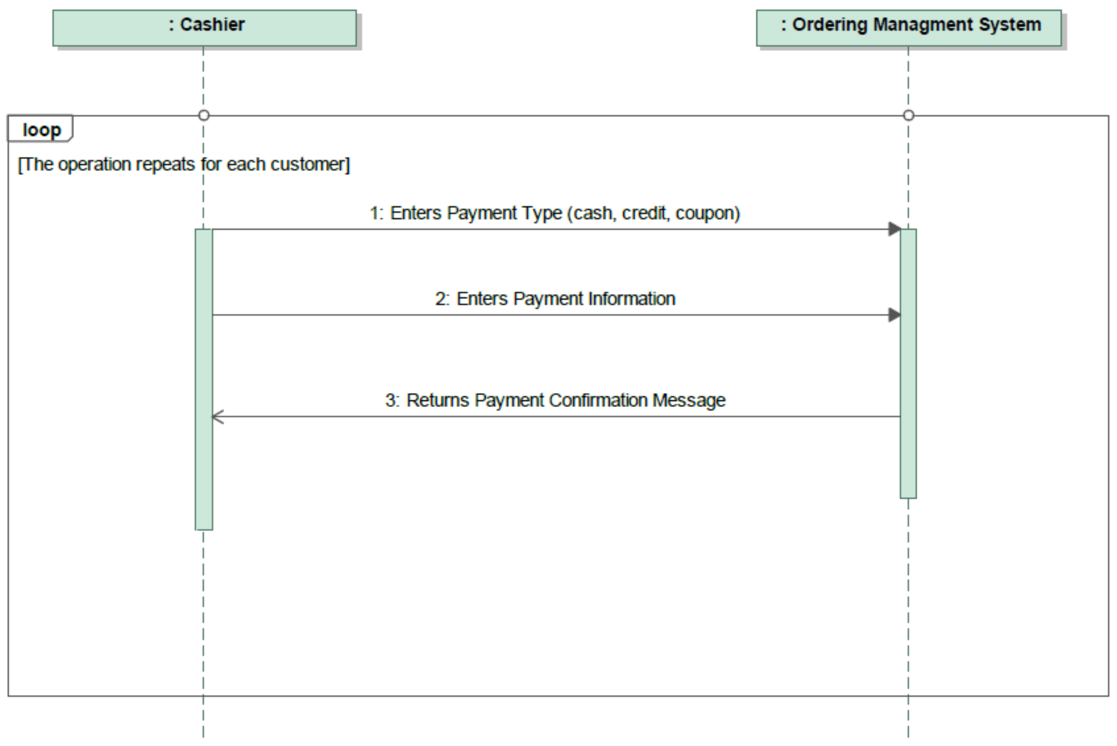
Use Case Diagram



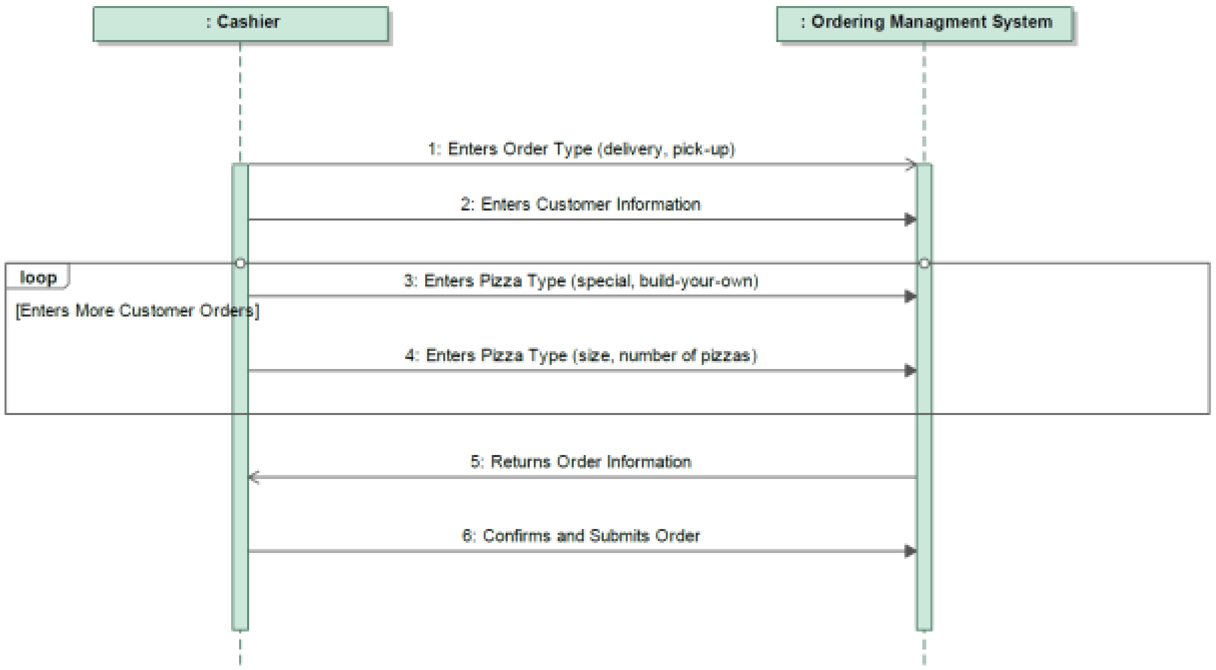
Domain Model



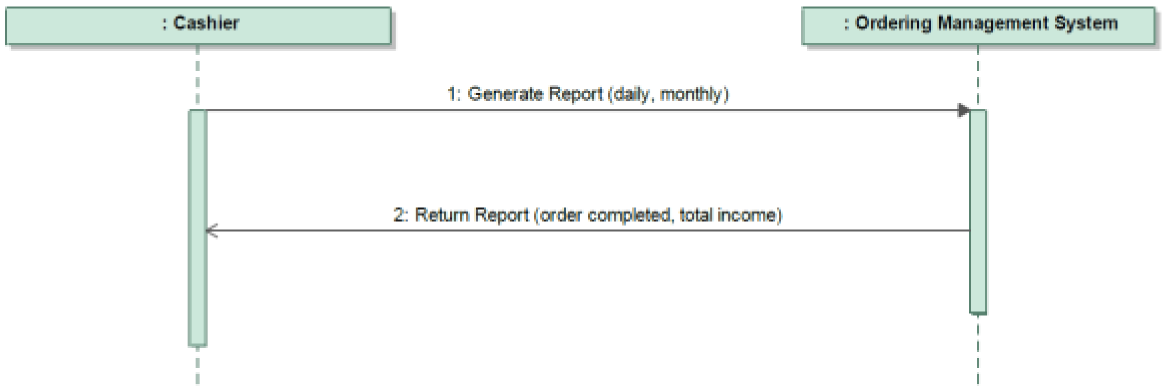
System Sequence Diagram (Payment)


: Cashier
: Ordering Managment System
loop
[The operation repeats for each customer]
1: Enters Payment Type (cash, credit, coupon)
2: Enters Payment Information
3: Returns Payment Confirmation Message

System Sequence Diagram (Order)


: Cashier
: Ordering Managment System
1: Enters Order Type (delivery, pick-up)
2: Enters Customer Information
loop
[Enters More Customer Orders]
3: Enters Pizza Type (special, build-your-own)
4: Enters Pizza Type (size, number of pizzas)
5: Returns Order Information
6: Confirms and Submits Order

System Sequence Diagram (Report)


: Cashier
: Ordering Management System
1: Generate Report (daily, monthly)
2: Return Report (order completed, total income)

Payment Use Case

Scope: Pizza Ordering System

Level: User Goal

Primary Actor: Cashier

Stakeholders and Interests

- Cashier: Wants accurate, fast entry and no payment errors. Has option to cancel order at any time during the transaction. Cashier also wants to have access to produce reports.

- IT/Development Team: Wants to ensure that the pizza ordering system functions from both front-and back-end perspectives.

Pre-Conditions: Cashier must be logged in to system.

Post Conditions: Kitchen cooks food according to cashier's order specifications. Each transaction/sale is saved. Tax is correctly calculated. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario:

1. System returns the order information to cashier.
2. Cashier enters cash option.
3. Cashier inputs amount of cash tendered by customer.
4. System returns cash amount owed to customer.
5. System returns return payment confirmation message.

Extensions:

2a. Cashier inputs credit card information.

   1. Cashier skips steps 3-4.

2b. Cashier inputs coupon information.

   1. Cashier skips steps 3-4.

3a. Cashier inputs insufficient cash funds.

   1. System returns message stating insufficient cash received.
   2. Cashier must receive sufficient cash funds to progress to step 4.

4a. Cashier received exact cash amount for order.

   1. Cashier returns zero cash to customer.

Special Requirements:

We want a system that is easy to use for individuals from any educational background. We want the pizza ordering system to be self-evident and simple.

Frequency of Occurrence: Could be nearly continuous

<u>Order Use Case</u>

Scope: Pizza Ordering System

Level: User Goal

Primary Actor: Cashier

Stakeholders and Interests

- Cashier: Wants accurate, fast entry and no payment errors. Has option to cancel order at any time during the transaction. Cashier also wants to have access to produce reports.

- IT/Development Team: Wants to ensure that the pizza ordering system functions from both front-and back-end perspectives.

Pre-Conditions: Cashier must be logged in to system.

Post Conditions: Kitchen cooks food according to cashier's order specifications. Each transaction/sale is saved. Tax is correctly calculated. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario:

1. Cashier inputs customer delivery.
2. Cashier enters Old Customer option.
3. Cashier enters Old Customer's phone-number.
4. System returns Old Customer's delivery information.
5. Cashier inputs custom-made AAA pizza.
6. Cashier inputs size and number of pizzas.
7. Cashier inputs check-out and finishes order.
8. Cashier enter <u>payment</u>.
9. System returns order confirmation information.
10. Cashier confirms and submits final order.

Extensions:

1a. Cashier inputs delivery.

1. Cashier enters customer’s name.
2. Cashier skips steps 2-4.

2a. Cashier inputs new customer.

1. Cashier enters customer’s address and phone number.
2. Cashier skips steps 3-4.

5a. Cashier inputs Philly Cheese Steak pizza.

7a. Cashier inputs more orders before submitting final order.

1. Cashier repeats steps 5-6.

Special Requirements:

We want a system that is easy to use for individuals from any educational background- we want the pizza ordering system to be self-evident and simple.

Frequency of Occurrence: Could be nearly continuous

<u>Report Use Case</u>

Scope: Pizza Ordering System

Level: User Goal

Primary Actor: Cashier

Stakeholders and Interests

- Cashier: Wants accurate, fast entry and no payment errors. Has option to cancel the order at any time during the transaction. Cashier also wants to have access to produce reports. Cashier also wants option to cancel order at any time.

- IT/Development Team: Wants to ensure that the pizza ordering system functions well from both front-and back-end perspectives.

Pre-Conditions: Cashier must be logged in to system.

Post Conditions: Kitchen cooks food according to cashier's order specifications. Each transaction/sale is saved. Tax is correctly calculated. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario:

1. Cashier submits request to generate report.
2. Cashier inputs daily report.
3. Cashier enters month, day, and year.
4. System returns number of orders completed and total income for specified day.

Extensions:

2a. Cashier inputs monthly report.

3a. Cashier inputs month and year.

4a. System returns number of orders completed and total income for designated month.

Special Requirements:

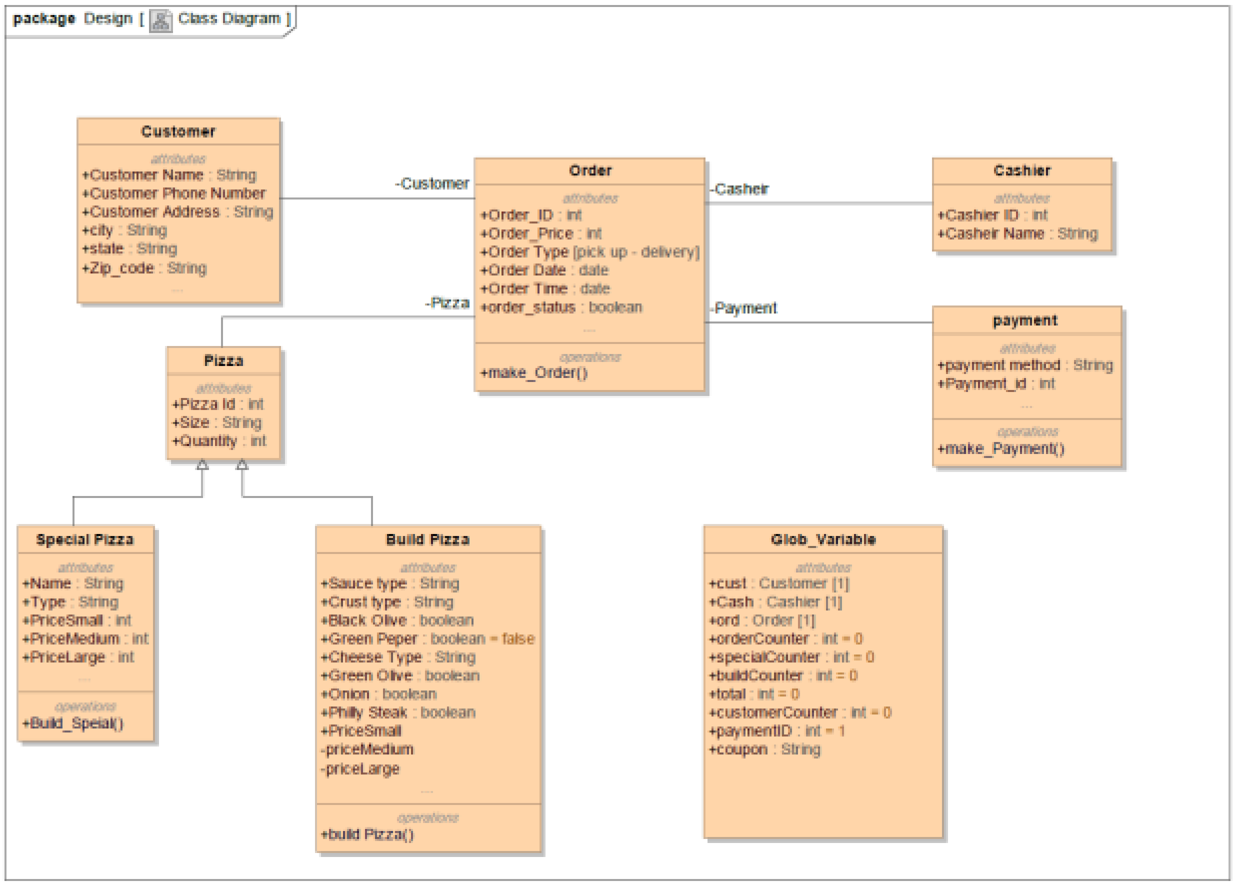
We want a system that is easy to use for individuals from any educational background. We want the pizza ordering system to be self-evident and simple.

Frequency of Occurrence: Could be nearly continuous

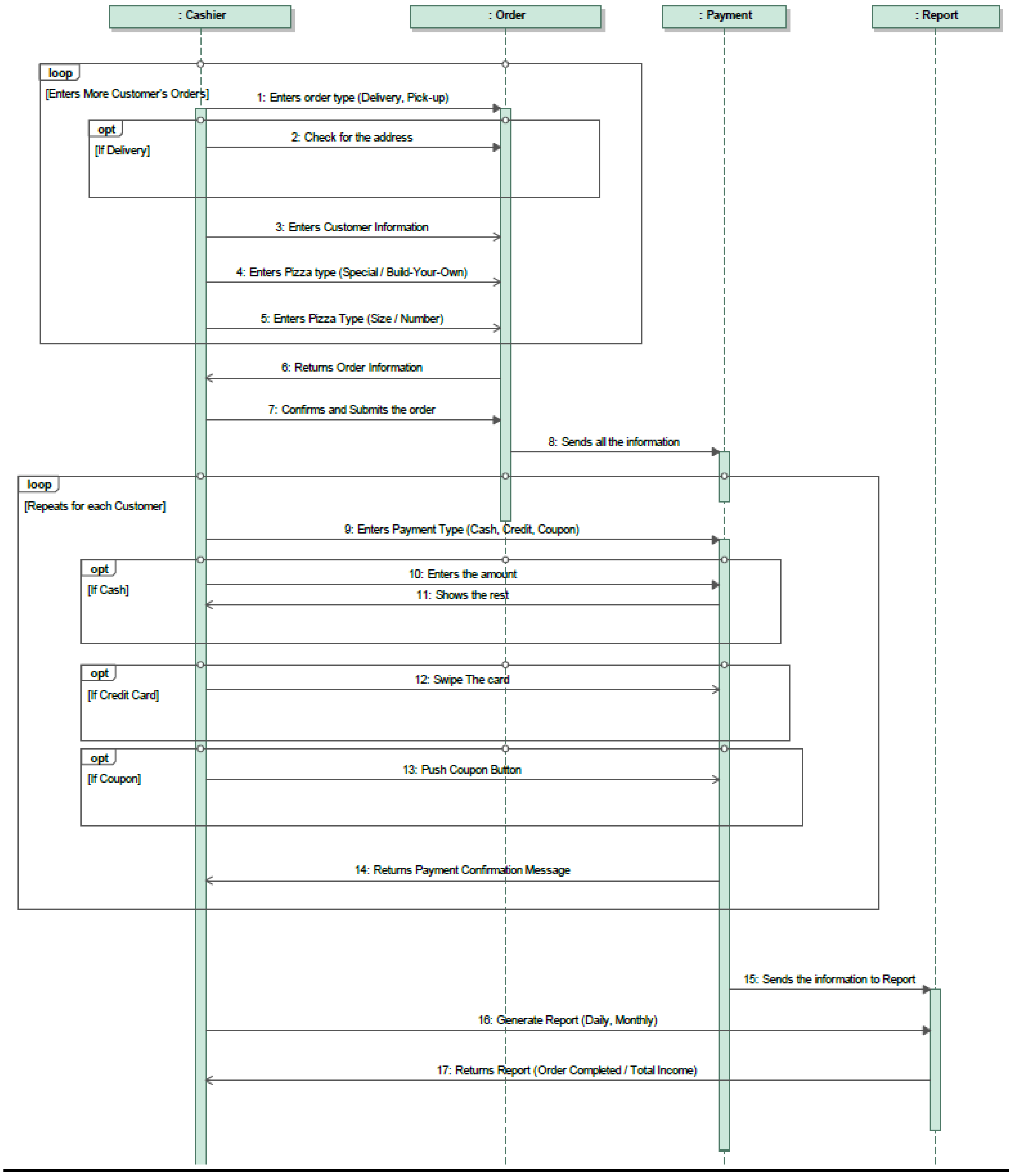
Design Artifacts

The Design Artifacts include the Class Diagram and the Interaction Diagram.

Class Diagram



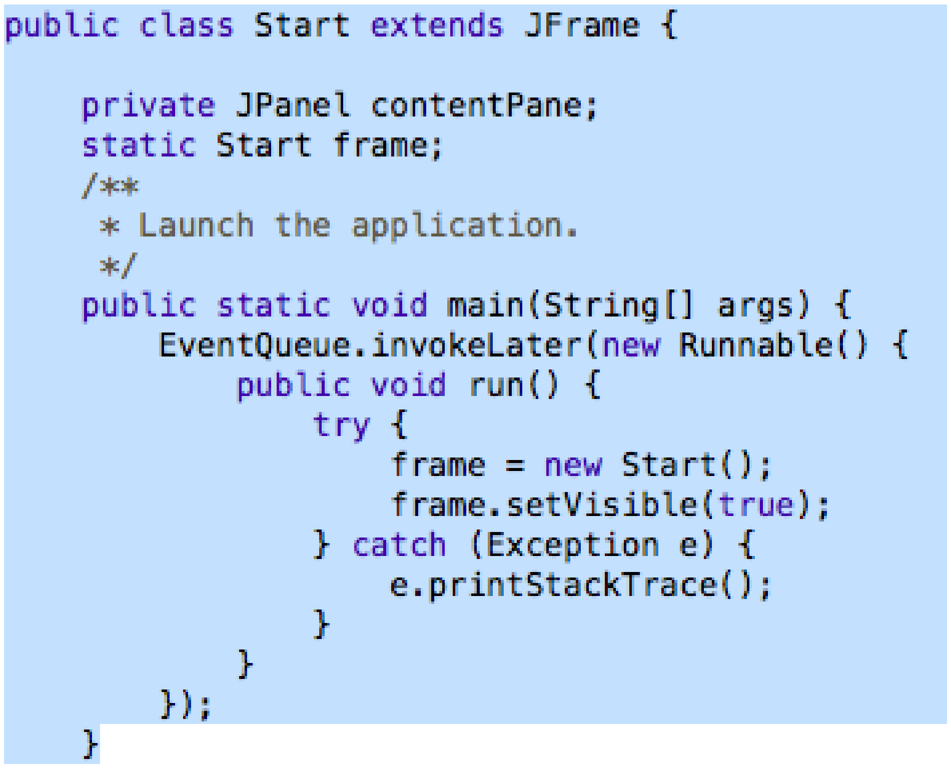
<u>Interaction Diagram</u>



: Cashier
: Order
: Payment
: Report
loop
[Enters More Customer's Orders]
1: Enters order type (Delivery, Pick-up)
opt
[If Delivery]
2: Check for the address
3: Enters Customer Information
4: Enters Pizza type (Special / Build-Your-Own)
5: Enters Pizza Type (Size / Number)
6: Returns Order Information
7: Confirms and Submits the order
8: Sends all the information
loop
[Repeats for each Customer]
9: Enters Payment Type (Cash, Credit, Coupon)
opt
[If Cash]
10: Enters the amount
11: Shows the rest
opt
[If Credit Card]
12: Swipe The card
opt
[If Coupon]
13: Push Coupon Button
14: Returns Payment Confirmation Message
15: Sends the information to Report
16: Generate Report (Daily, Monthly)
17: Returns Report (Order Completed / Total Income)

Implementation Image/Code Sample

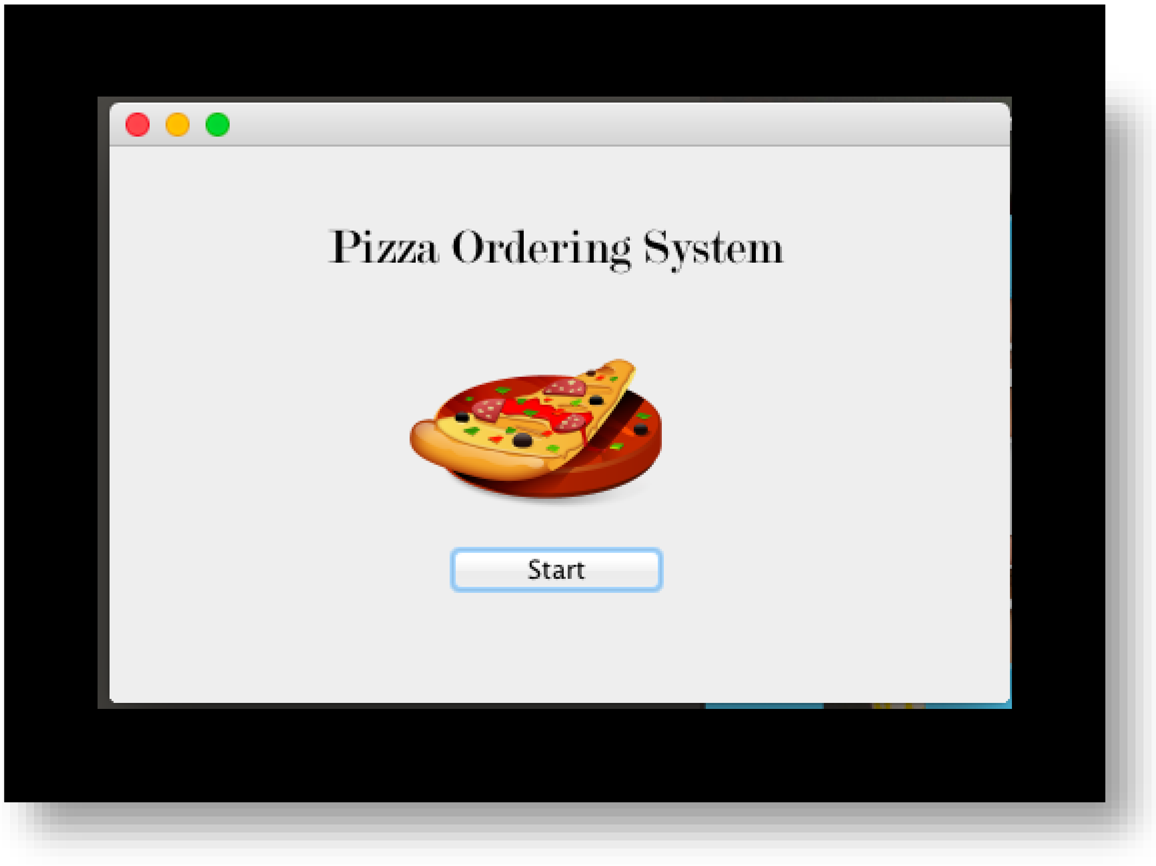
We emailed the Java source code to Dr. Rubin. Here we have included some code that implemented the use cases. The cashier was a rather easy class to implement and simply required defining the attributes/variables for the class. Coding the “Start” class was, as the diagrams imply, much more involved than ones like the cashier. Here, we used Java Swing to generate a GUI for the initial POS interaction, as the sample shows:



```
public class Start extends JFrame {

    private JPanel contentPane;
    static Start frame;
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    frame = new Start();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
```

The class ultimately generates the following image:



Organization and Communication

In terms of organization, each member participated in Analysis, Design, and Coding. We first focused on the system requirements, ensuring that each participant understood their respective roles and responsibilities. First, we started with the Analysis phase, which included the Use Case Diagram, the Domain Model, the Sequence Diagrams, and Fully-Dressed Use Cases. Simultaneously, we began to code our project in Java – a language with which we all had basic knowledge. As we worked on coding, we initiated the Design phase, composing documents such as the Interaction Diagram and the Class Diagram. In tandem, we finished a functioning Java code application.

In terms of communication, we communicated through Google Hangout, email, text, and in person, both during team-time and outside of class.

Lessons Learned

Our team followed Dr. Rubin's advice: start early! This was a key factor in our success. We maintained clear, consistent communication to facilitate progress with the project. Also, if we had any conflict, misunderstandings, or general questions, we contacted Dr. Rubin for feedback. This last step was crucial, as it ensured that our project stayed on track without generating team animosity. Significantly, this project strengthened the rudimentary knowledge of Java and Object Oriented principles we gained in SEIS 601. Each member of our team confidently left the project ready to move to more intermediate and advanced Java concepts.