Group 7

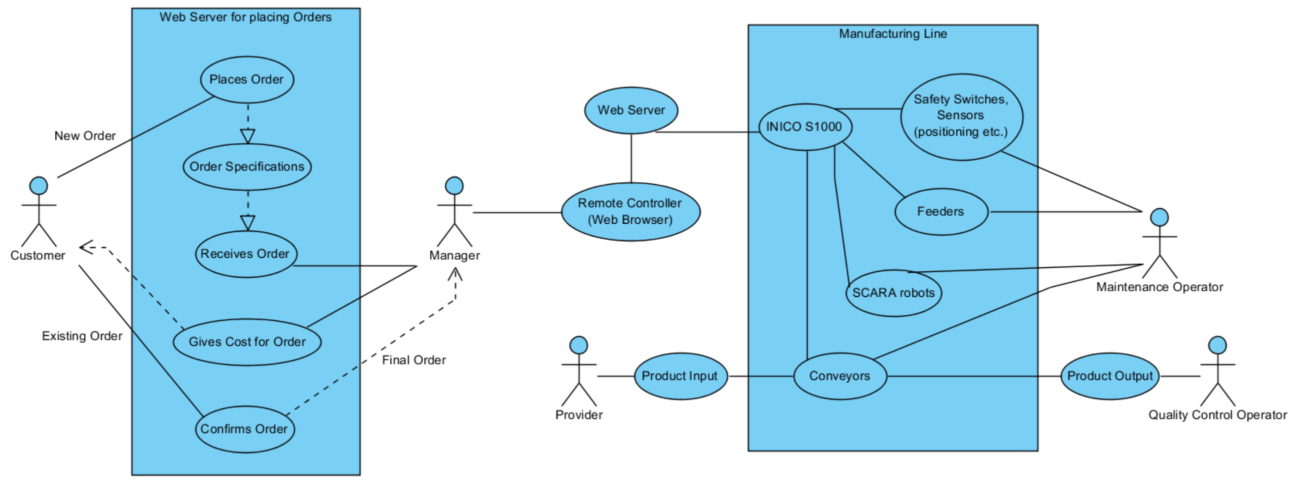
Carlos Guttormsen
Miska Örmälä
Yuan Qi

Third Iteration Report

This is group seven's third iteration report for the Introduction to Industrial Informatics course work. In this report we refined our project work's use cases further to take in consideration the customer. We also produced the first prototype of our source code and developed state charts for the classes in the code to further show the thought process behind each class.

Use Case Diagram

In this iteration we went over our use cases for the course project and refined them further. We took into consideration the customer's role and how it would get its order placed. With the refined use cases we then produced an updated use case diagram which is shown in the picture 1 below.



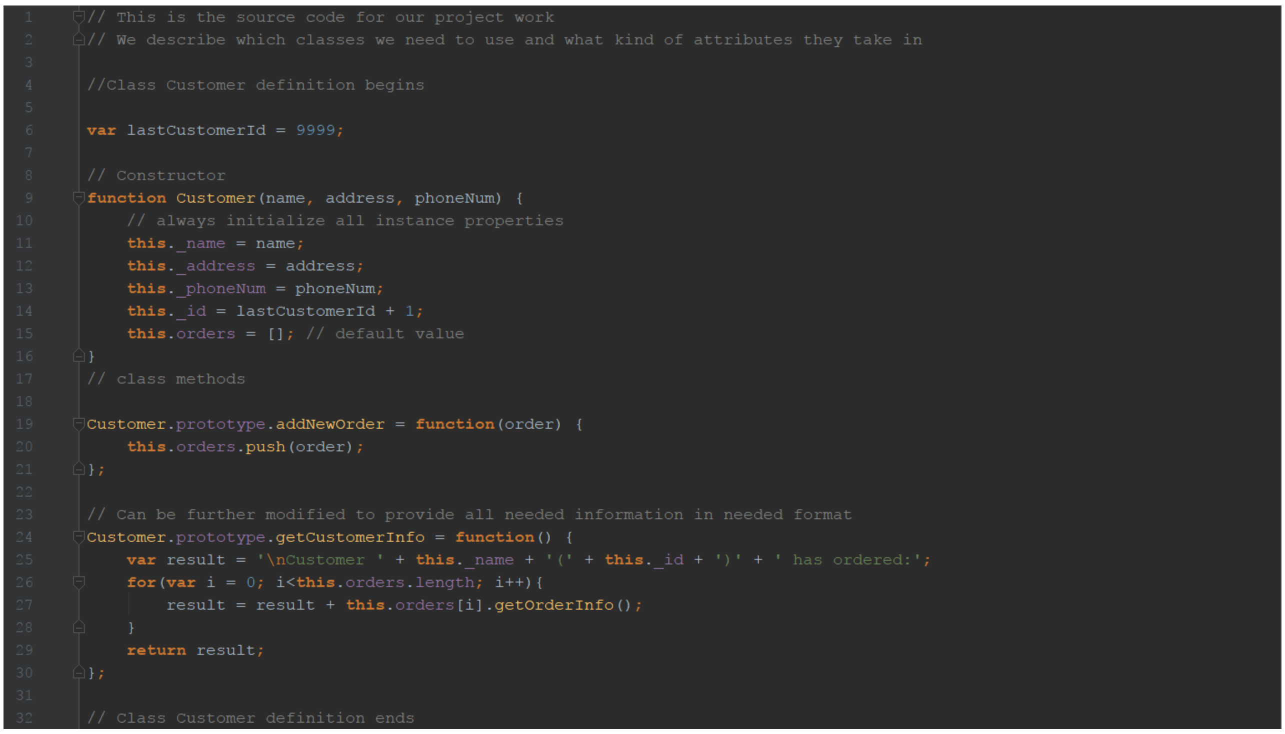
Picture 1. Use Case Diagram

This use case diagram consists of two systems; the web server in which the customer places its order and the manufacturing line which then produces the placed order accordingly. As shown in the UCD, the manager acts as the middleman between the two systems.

Source Code and Related Charts

To start forming the backbone of our project work's so-called user interface where our customer would be able to place its order, we produced the first prototype of our source code. This source code consists three classes which are used as a guidance to how an order placed by a customer could be processed in our program. All the information regarding to our customer and its order would be placed by the customer on our web server (picture 1) which is still being developed.

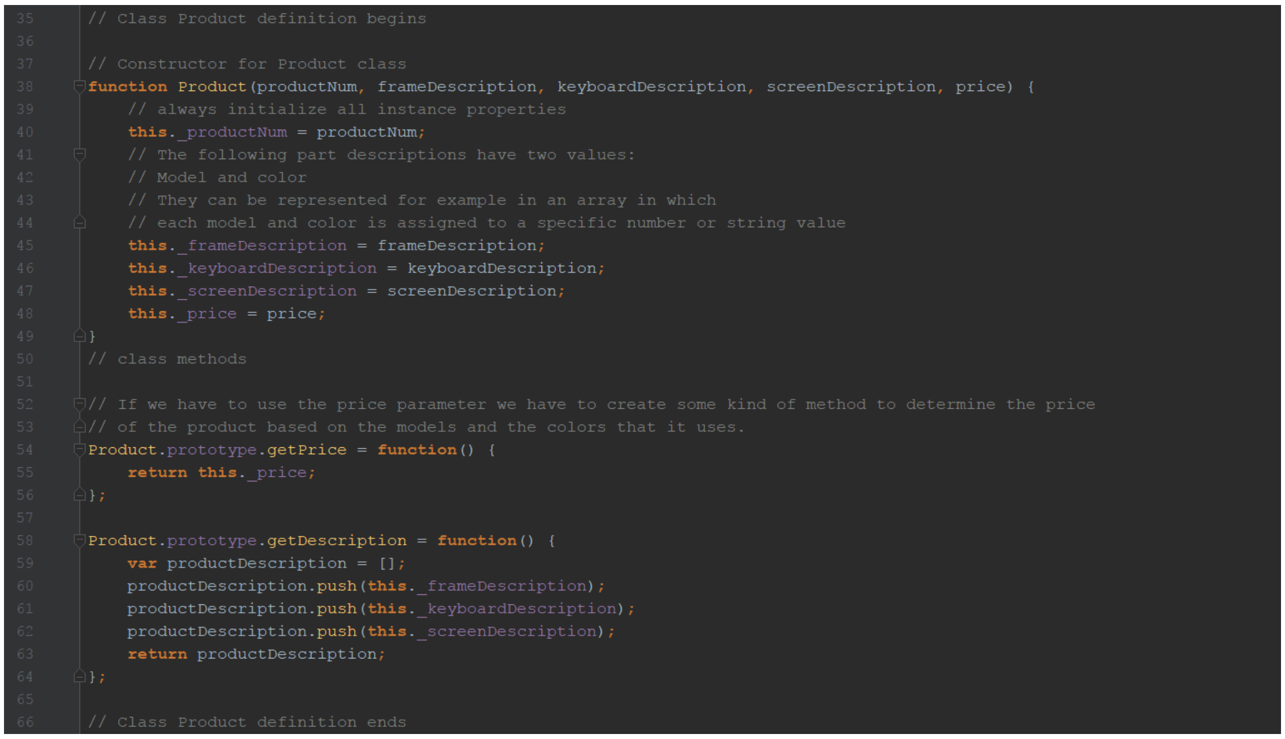
The three classes mentioned above are as follows: Customer, Product, Order. These are represented in the following pictures 2-4.



```
// This is the source code for our project work
// We describe which classes we need to use and what kind of attributes they take in

//Class Customer definition begins

var lastCustomerId = 9999;

// Constructor
function Customer(name, address, phoneNum) {
    // always initialize all instance properties
    this._name = name;
    this._address = address;
    this._phoneNum = phoneNum;
    this._id = lastCustomerId + 1;
    this.orders = []; // default value
}
// class methods

Customer.prototype.addNewOrder = function(order) {
    this.orders.push(order);
};

// Can be further modified to provide all needed information in needed format
Customer.prototype.getCustomerInfo = function() {
    var result = '\nCustomer ' + this._name + '(' + this._id + ')' + ' has ordered:';
    for(var i = 0; i<this.orders.length; i++){
        result = result + this.orders[i].getOrderInfo();
    }
    return result;
};

// Class Customer definition ends
```

Picture 2. Customer Class

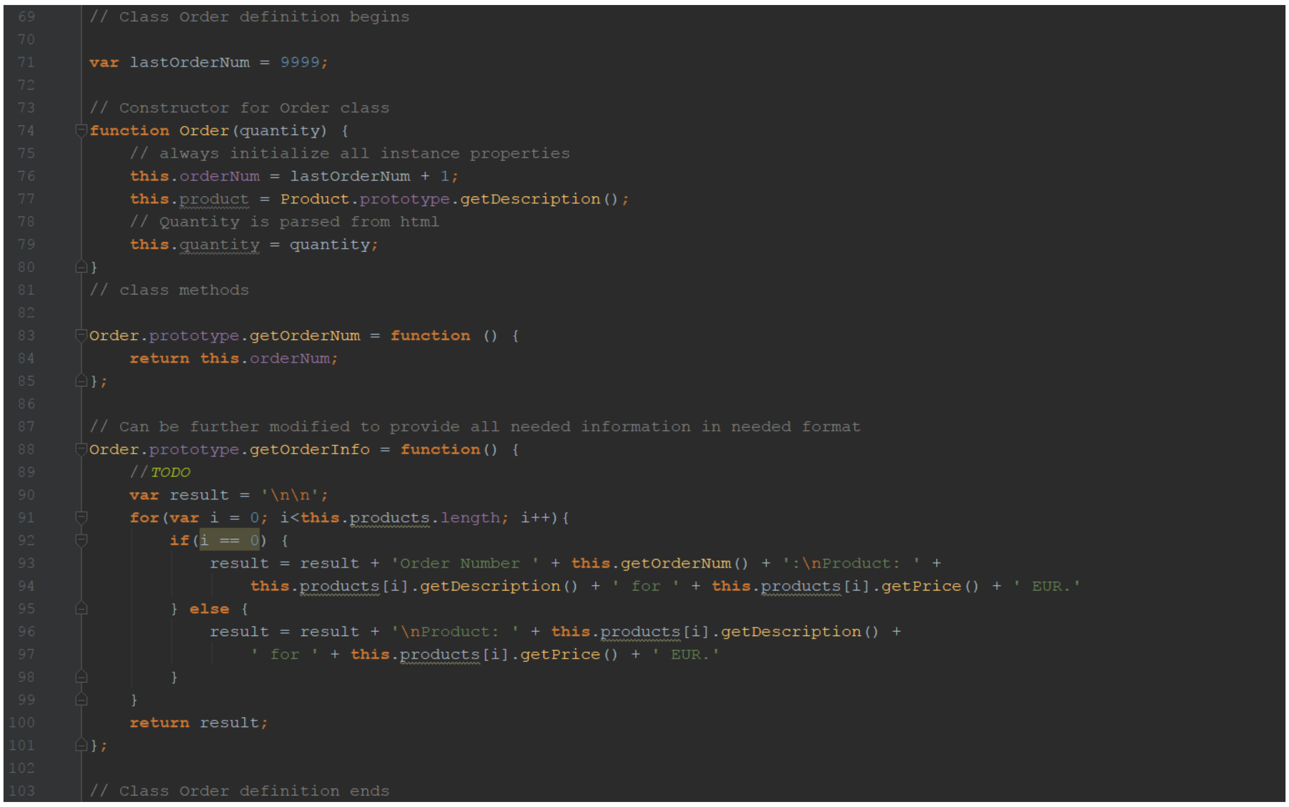
This is an early idea of what our customer class could look like.



```
// Class Product definition begins

// Constructor for Product class
function Product(productNum, frameDescription, keyboardDescription, screenDescription, price) {
    // always initialize all instance properties
    this._productNum = productNum;
    // The following part descriptions have two values:
    // Model and color
    // They can be represented for example in an array in which
    // each model and color is assigned to a specific number or string value
    this._frameDescription = frameDescription;
    this._keyboardDescription = keyboardDescription;
    this._screenDescription = screenDescription;
    this._price = price;
}
// class methods

// If we have to use the price parameter we have to create some kind of method to determine the price
// of the product based on the models and the colors that it uses.
Product.prototype.getPrice = function() {
    return this._price;
};

Product.prototype.getDescription = function() {
    var productDescription = [];
    productDescription.push(this._frameDescription);
    productDescription.push(this._keyboardDescription);
    productDescription.push(this._screenDescription);
    return productDescription;
};

// Class Product definition ends
```

Picture 3. Product Class

This is an early idea of what our product class could look like.

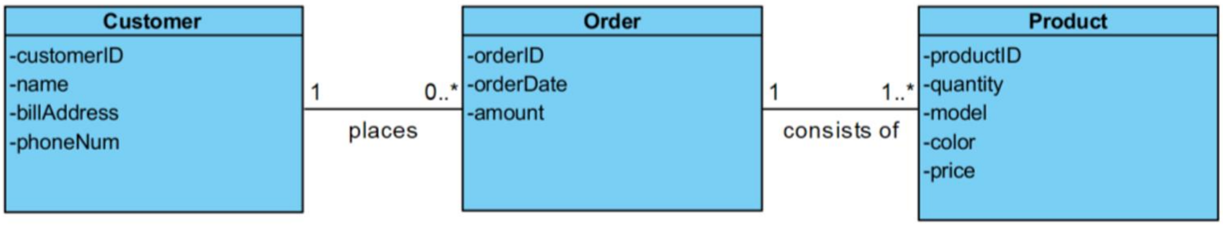


```
// Class Order definition begins

var lastOrderNum = 9999;

// Constructor for Order class
function Order(quantity) {
    // always initialize all instance properties
    this.orderNum = lastOrderNum + 1;
    this.product = Product.prototype.getDescription();
    // Quantity is parsed from html
    this.quantity = quantity;
}
// class methods

Order.prototype.getOrderNum = function () {
    return this.orderNum;
};

// Can be further modified to provide all needed information in needed format
Order.prototype.getOrderInfo = function() {
    //TODO
    var result = '\n\n';
    for(var i = 0; i<this.products.length; i++){
        if(i == 0) {
            result = result + 'Order Number ' + this.getOrderNum() + ':\nProduct: ' +
                this.products[i].getDescription() + ' for ' + this.products[i].getPrice() + ' EUR.'
        } else {
            result = result + '\nProduct: ' + this.products[i].getDescription() +
                ' for ' + this.products[i].getPrice() + ' EUR.'
        }
    }
    return result;
};

// Class Order definition ends
```

Picture 4. Order Class

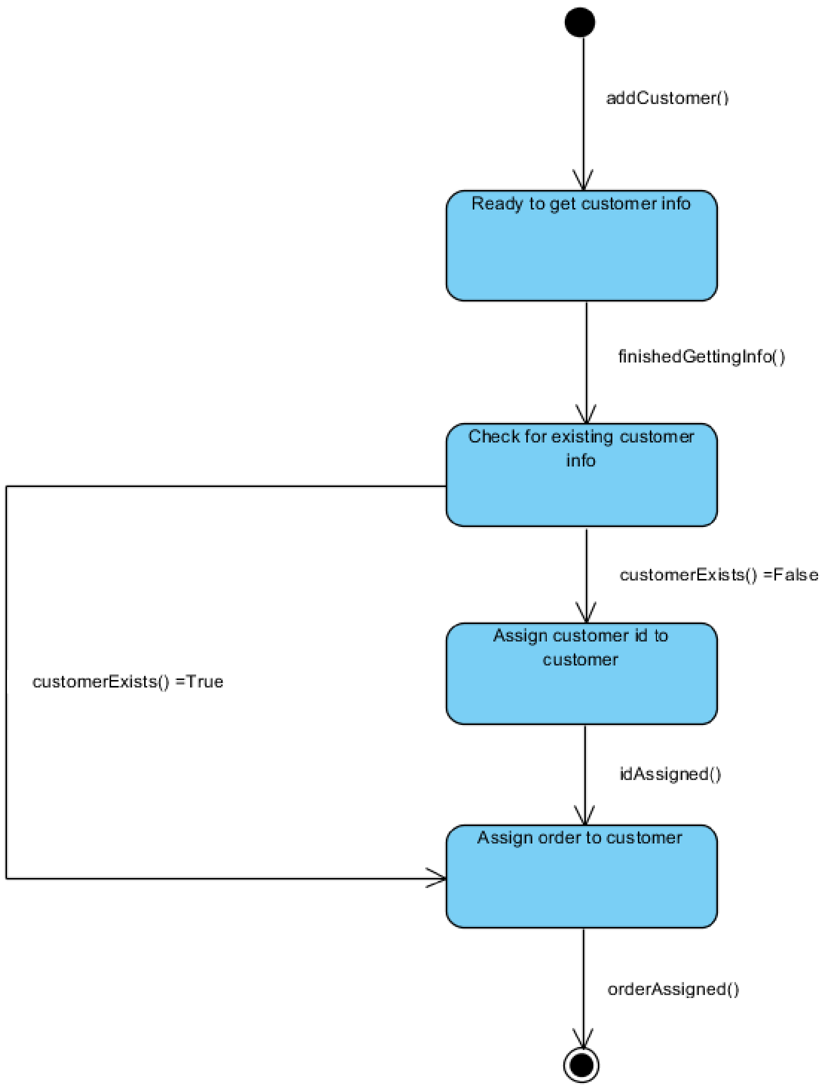
This is an early idea of what our order class could look like.

In our last iteration we created a class diagram which summarizes the idea behind the classes of the source code above in a brief manor. This class diagram is illustrated in the picture 5 below.

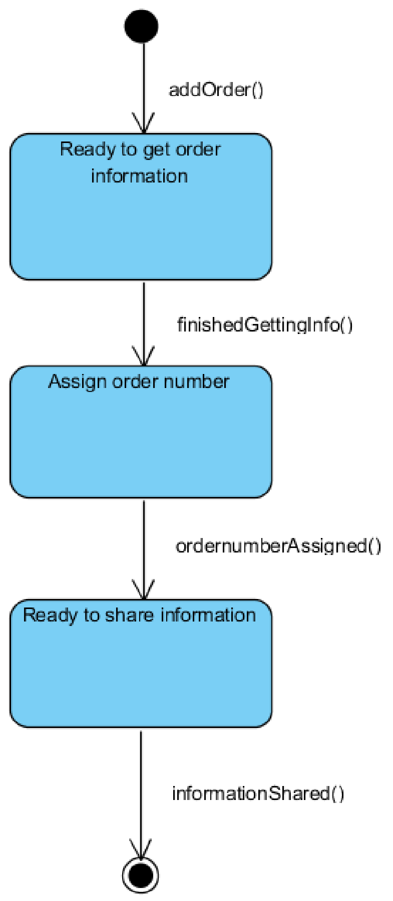


Picture 5. Class diagram

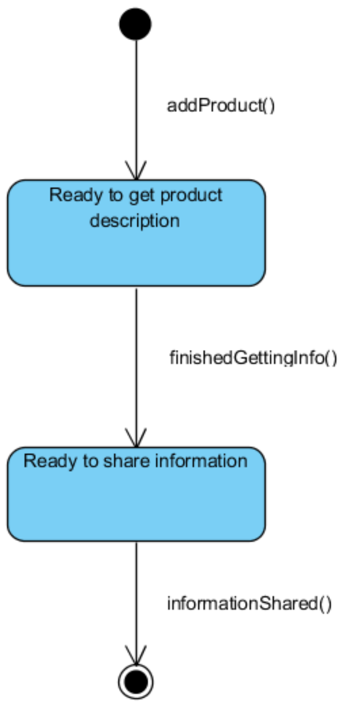
Finally, we developed the required state charts for the classes used in our source code and in our class diagram. In these we demonstrate the thought process behind each action and the logic they follow. In other words, we describe the different states and the exit conditions for them. The state charts for each class are represented in the following pictures 6-8.



Picture 6. Customer State Chart



Picture 7. Order State Chart



Picture 8. Product State Chart