DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COEN 6312 – Model Driven Software Engineering
Winter 2022

Report - Deliverable 2
on

E-COMMERCE SYSTEM

Course Instructor

DR. WAHAB HAMOU-LHADJ

Submitted By

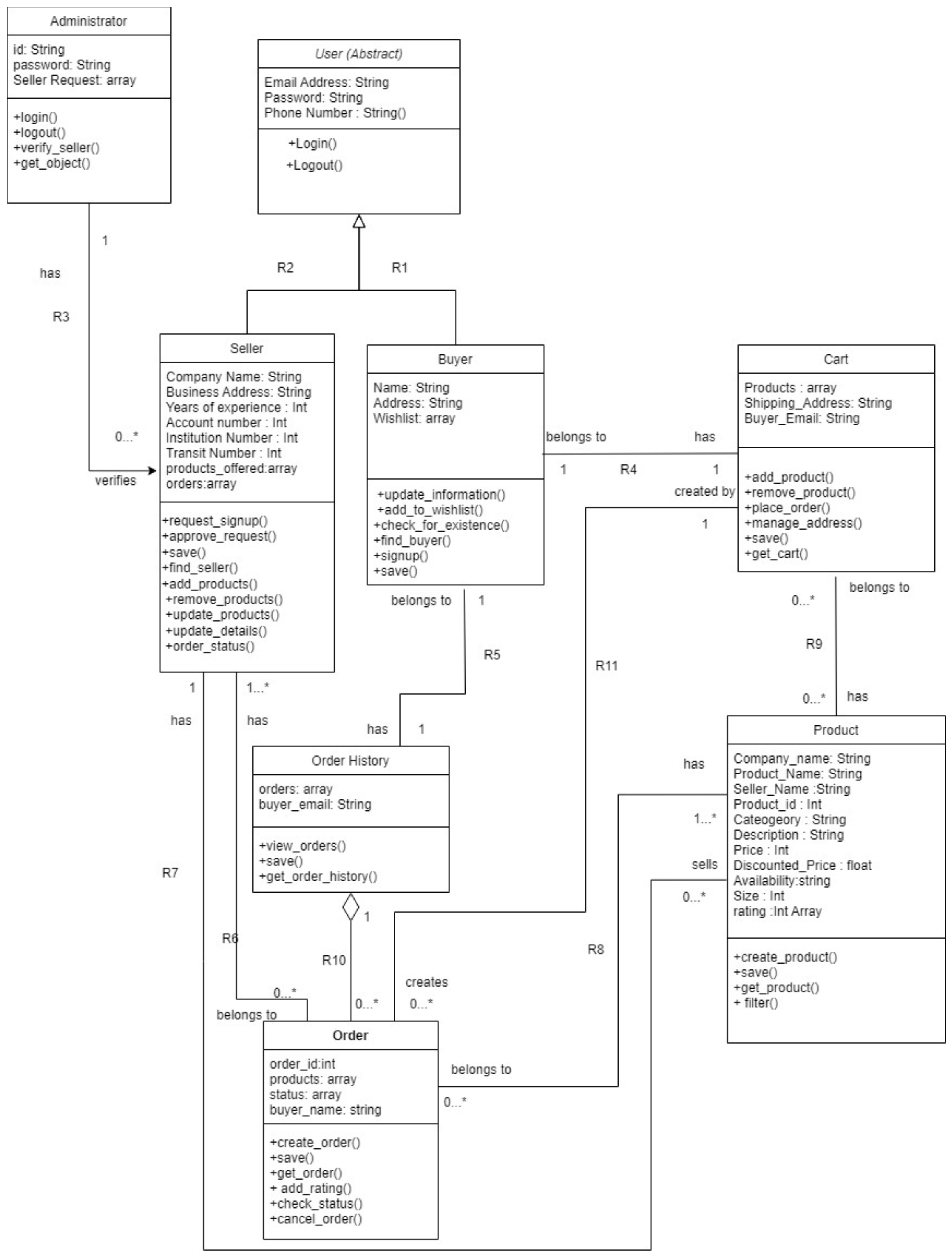
GURVEEN KAUR BAWEJA (40164885)
KARANJOT SINGH KOCHAR (40161109)
NAVDEEP SINGH (40161110)
SRI HARSHA VELLATURI (40185295)
SRINATH ANANTHULA (40174198)
PARUL MAINI (40181624)

Table of Contents

List of figures

Updated class diagram

The following figure represents the updated class diagram of our E-commerce system. The class diagram depicts the interactions between the classes and the methods which are implemented in programming.



OCL Rules

1. Any two buyers should have different email addresses and phone numbers.
   - ➔ Context: Buyer
     inv: allInstances -> forAll (b1, b2 | b1<>b2 implies b1.email_address <> b2.email_address AND b1.phone_number <> b2.phone_number)

2. Any two sellers should have different email addresses, company name, business addresses and phone numbers.
   - ➔ Context: Seller
     inv: allInstances -> forAll (s1, s2 | s1<>s2 implies s1.email_address <> s2.email_address
     AND
     s1.phone_number<>s2.phone_number
     AND
     s1.company_name<>s2.company_name
     AND
     s1.business_address<>s2.business_address

3. Buyer can add up to five items in the cart.
   - ➔ Context: Cart
     self. products -> size() <=5

4. Buyer can add up to ten items in the wishlist.
   - ➔ Context: Buyer
     Self. Wishlist -> size() <=10

5. Buyer cannot add items in the cart if out of stock.
   - ➔ Context: Cart:: add_product (p: Product)
     Inv:
     pre condition :
     self. Products -> excludes (p)
     p. availability -> 'in stock'
     post condition:
     self. Products -> includes(p)

6. No two products should have same product Id.
   - ➔ Context: product
     inv: allInstances -> forAll (p1, p2 | p1<>p2 implies p1.product_id <> p2.product_id)

7. Seller should be able to add a product which is already not being offered by that seller.
   - ➔ Context: Seller:: add_product(p :Product)
     inv:
     pre condition
     self. products_offered -> excludes(p)
     post condition
     self. Products_offered -> includes(p)

8. Product that is already in the cart can only be removed.
   - ➔ Context : Cart :: remove_product (p : Product)
     inv:
     pre condition
     self. products -> includes(p)
     post condition
     self. products -> excludes(p)

9. Shipping address should be one of the addresses linked with buyer account.
   - ➔ Context : Buyer
     inv:
     self.address -> includes(self. cart. shipping_address)

10. Order history should include all the orders placed by the user.
    - ➔ Context : Buyer
      inv:
      self.OrderHistory.orders -> includesAll (self.cart.orders)

11. Cart should have at least 1 item before placing order and cart should be empty after placing the order.
    - ➔ Context : Cart :: place_order()
      inv : pre condition
      self.products -> size() >=1
      post condition
      self. products -> size() =0

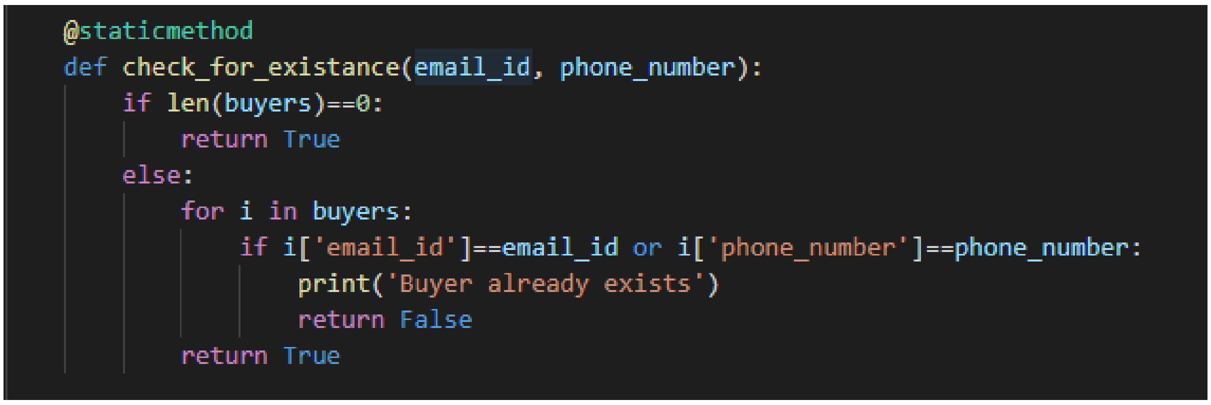
12. Order can be cancelled if its status is "in processing".
    - ➔ Context : Order :: cancel_order()
      inv :
      pre condition
      self. status = 'in processing '
      post condition
      self. status = 'cancelled'

13. Seller can update the status of only those orders which includes the products offered by that seller.
    - ➔ Context : Seller
      inv:
      self. product . order -> includesAll (self. order)

14. Product can be rated only after it is delivered.
    - ➔ Context : Order :: add_rating()
      inv :
      pre condition
      self. status= ‘delivered’

Implementation of OCL constraints

1. Any two buyers should have different email addresses and phone numbers.

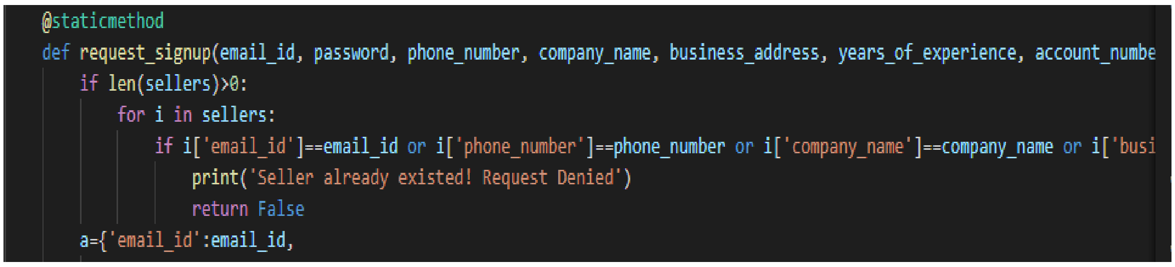


```python
@staticmethod
def check_for_existance(email_id, phone_number):
    if len(buyers)==0:
        return True
    else:
        for i in buyers:
            if i['email_id']==email_id or i['phone_number']==phone_number:
                print('Buyer already exists')
                return False
        return True
```

Fig 1 – Buyers should have different email addresses and phone numbers.

When a buyer signs up at the system then before signing up, system checks if any existing buyer has same email or phone number as entered by the new buyer. If yes, then it gives the error that buyer already exists and prevents the new buyer from signing up.

2. Any two sellers should have different email addresses, company name, business addresses and phone numbers.

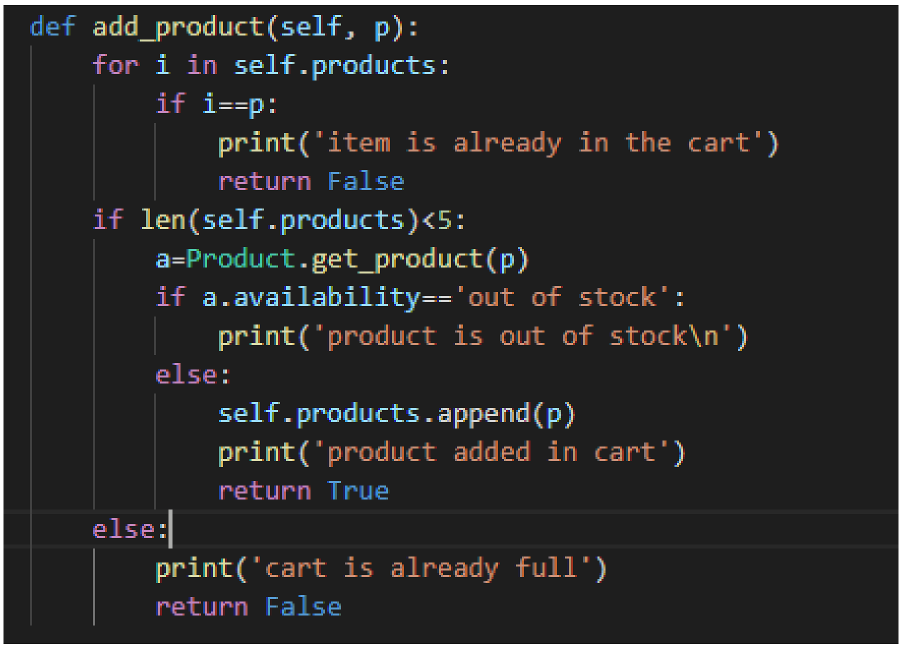


```python
@staticmethod
def request_signup(email_id, password, phone_number, company_name, business_address, years_of_experience, account_numbe
    if len(sellers)>0:
        for i in sellers:
            if i['email_id']==email_id or i['phone_number']==phone_number or i['company_name']==company_name or i['busi
                print('Seller already existed! Request Denied')
                return False
    a={'email_id':email_id,
```

Fig 2 – Sellers should have different email addresses, company name, business address and phone numbers.

When a seller signs up at the system then before signing up, system checks if any existing seller has same email, phone number, company name or business address as entered by the new seller. If yes, then it gives the error that seller already exists and prevents the new seller from signing up.

3. Buyer can add up to five items in the cart.

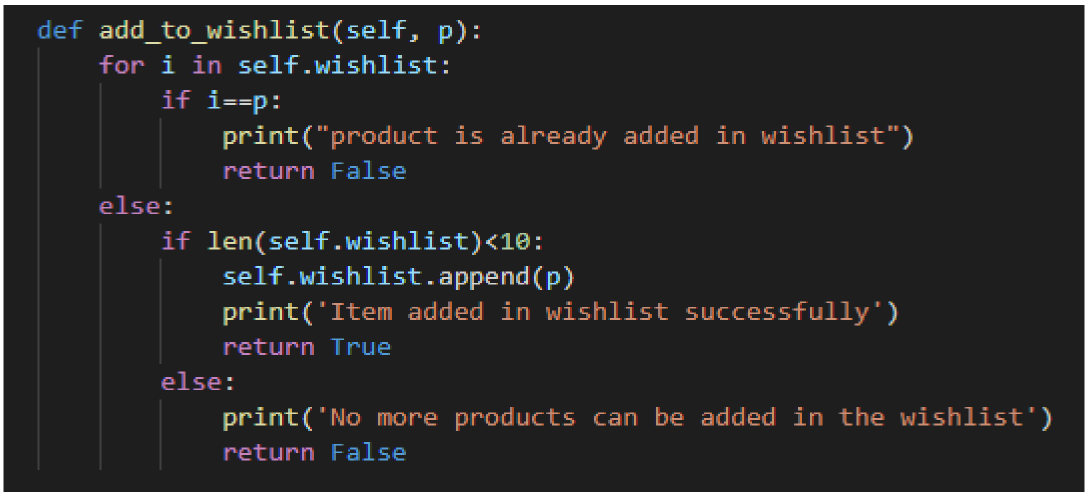


```python
def add_product(self, p):
    for i in self.products:
        if i==p:
            print('item is already in the cart')
            return False
    if len(self.products)<5:
        a=Product.get_product(p)
        if a.availability=='out of stock':
            print('product is out of stock\n')
        else:
            self.products.append(p)
            print('product added in cart')
            return True
    else:
        print('cart is already full')
        return False
```

Fig 3 – Buyers can add up to five products in the cart.

Before adding an item in the cart, first the size of the products array is checked, if it is less than 5 only then the item is added.

4. Buyer can add up to ten items in the Wishlist.

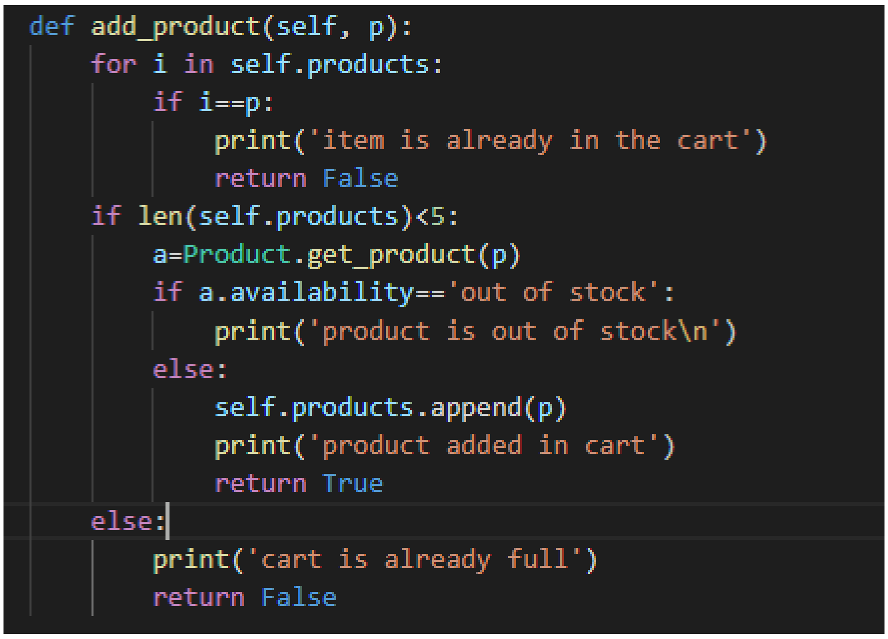


```python
def add_to_wishlist(self, p):
    for i in self.wishlist:
        if i==p:
            print("product is already added in wishlist")
            return False
    else:
        if len(self.wishlist)<10:
            self.wishlist.append(p)
            print('Item added in wishlist successfully')
            return True
        else:
            print('No more products can be added in the wishlist')
            return False
```

Fig 4 – Buyer can add up to 10 products into the Wishlist.

Before adding an item in the cart, first the size of the Wishlist array is checked, if it is less than 10 only then the item is added in Wishlist.

5. Buyer cannot add items in the cart if it is out of stock.

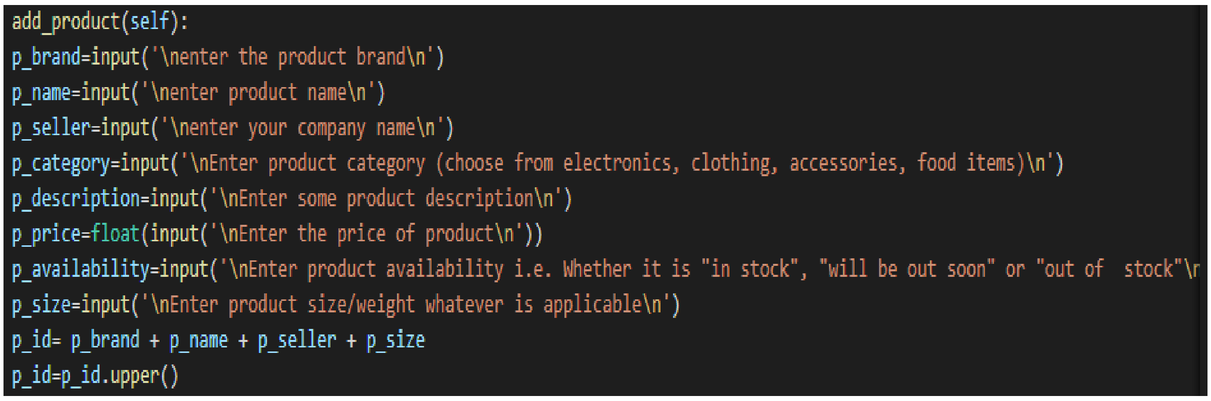


```
def add_product(self, p):
    for i in self.products:
        if i==p:
            print('item is already in the cart')
            return False
    if len(self.products)<5:
        a=Product.get_product(p)
        if a.availability=='out of stock':
            print('product is out of stock\n')
        else:
            self.products.append(p)
            print('product added in cart')
            return True
    else:
        print('cart is already full')
        return False
```

Fig 5 – Buyer can't add products into the cart which are out of stock.

Before adding an item in the cart, first the system checks the availability of the product. If it is out of stock then it will give the message that the product is out of stock and it will not be added in the cart.

6. No two products should have same product Id.

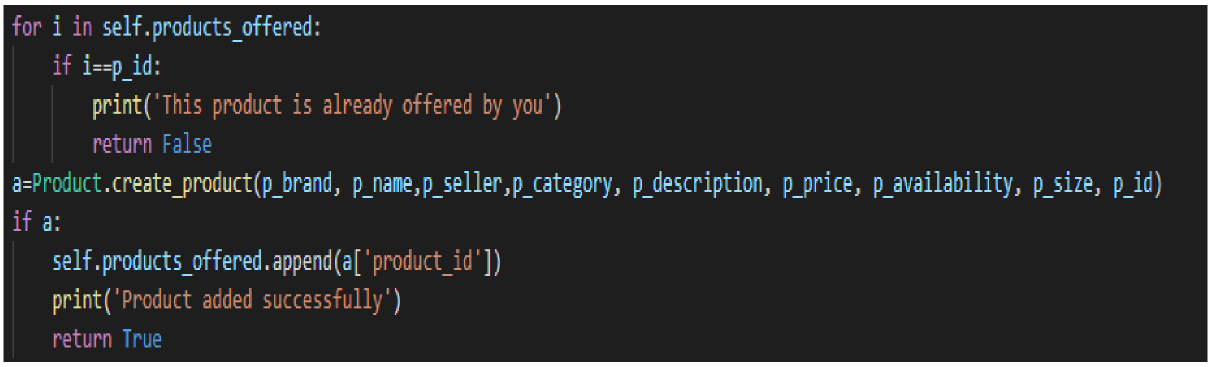


```
add_product(self):
p_brand=input('\nenter the product brand\n')
p_name=input('\nenter product name\n')
p_seller=input('\nenter your company name\n')
p_category=input('\nEnter product category (choose from electronics, clothing, accessories, food items)\n')
p_description=input('\nEnter some product description\n')
p_price=float(input('\nEnter the price of product\n'))
p_availability=input('\nEnter product availability i.e. Whether it is "in stock", "will be out soon" or "out of  stock"\r
p_size=input('\nEnter product size/weight whatever is applicable\n')
p_id= p_brand + p_name + p_seller + p_size
p_id=p_id.upper()
```

Fig 6 – Two products should not have the same product ID.

When a product is added by the seller, a unique product id is created using the combination of Brand name, product name, seller name and the size/weight information of the product and it is linked with each product. Before adding the product in the system, it first checks if any product is already existing with this product id or not. If yes, then it will give the message that product already exists.

7. Seller should be able to add a product which is already not being offered by that seller.

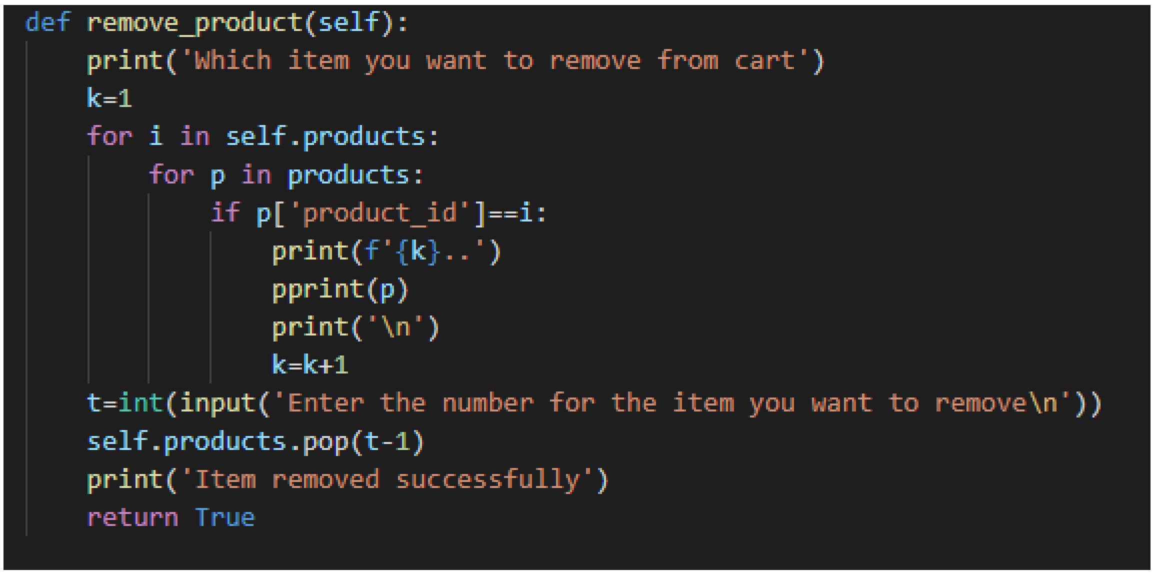


```python
for i in self.products_offered:
    if i==p_id:
        print('This product is already offered by you')
        return False
a=Product.create_product(p_brand, p_name,p_seller,p_category, p_description, p_price, p_availability, p_size, p_id)
if a:
    self.products_offered.append(a['product_id'])
    print('Product added successfully')
    return True
```

Fig 7 – Seller can only add new product other than existing product.

Before adding the product in the system, it first checks if any product is already existing with this product id or not. If yes, then it will give the message that product already exists.

8. Product that is already in the cart can only be removed.

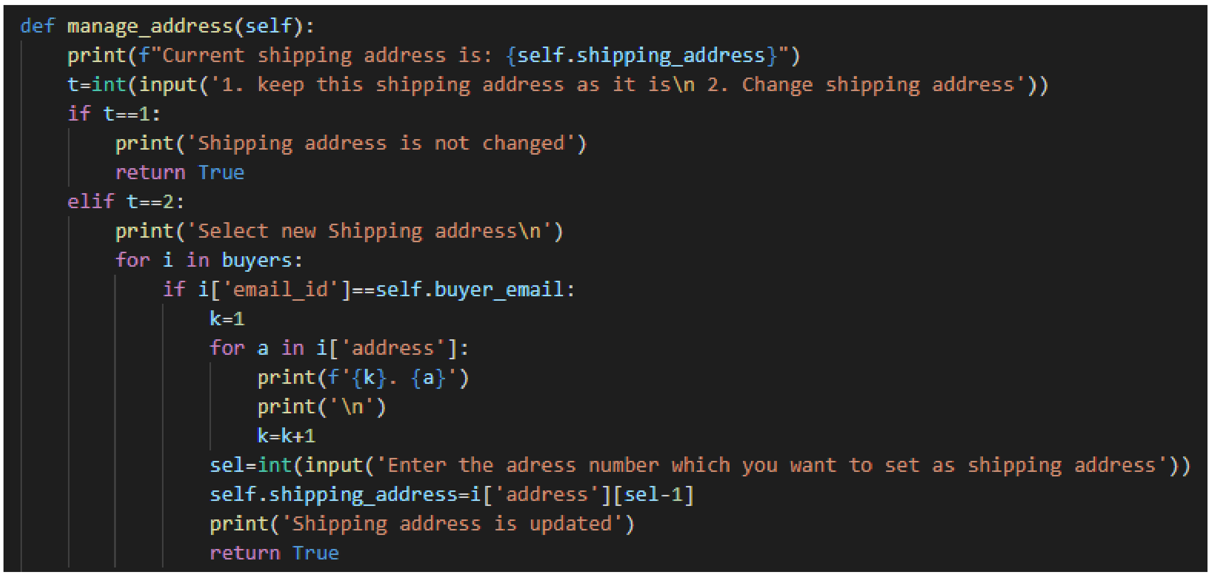


```python
def remove_product(self):
    print('Which item you want to remove from cart')
    k=1
    for i in self.products:
        for p in products:
            if p['product_id']==i:
                print(f'{k}..')
                pprint(p)
                print('\n')
                k=k+1
    t=int(input('Enter the number for the item you want to remove\n'))
    self.products.pop(t-1)
    print('Item removed successfully')
    return True
```

Fig 8 – Product that is in the cart can be removed.

When a buyer wants to remove a product from the cart, system will first show the products which are in the cart and take input from buyer that which product buyer wants to remove. Thus, ensuring that the product which is being removed is present in the cart.

9. Shipping address should be one of the addresses linked with buyer account.

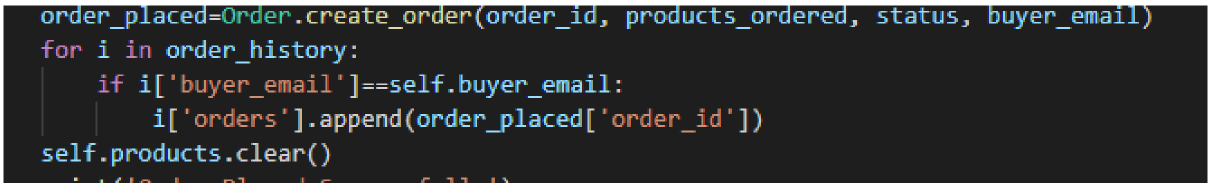


```python
def manage_address(self):
    print(f"Current shipping address is: {self.shipping_address}")
    t=int(input('1. keep this shipping address as it is\n 2. Change shipping address'))
    if t==1:
        print('Shipping address is not changed')
        return True
    elif t==2:
        print('Select new Shipping address\n')
        for i in buyers:
            if i['email_id']==self.buyer_email:
                k=1
                for a in i['address']:
                    print(f'{k}. {a}')
                    print('\n')
                    k=k+1
                sel=int(input('Enter the adress number which you want to set as shipping address'))
                self.shipping_address=i['address'][sel-1]
                print('Shipping address is updated')
                return True
```

Fig 9 – Shipping address should be linked with the buyer account.

When Buyer tries to change the shipping address, the system will show all the addresses linked with the buyer's account. Then it will take input from the buyer that which address buyer wants to set as the shipping address. Therefore, the constraint that the shipping address should be one of the addresses linked with buyer's account will be satisfied.

10. Order history should include all the orders placed by the user.



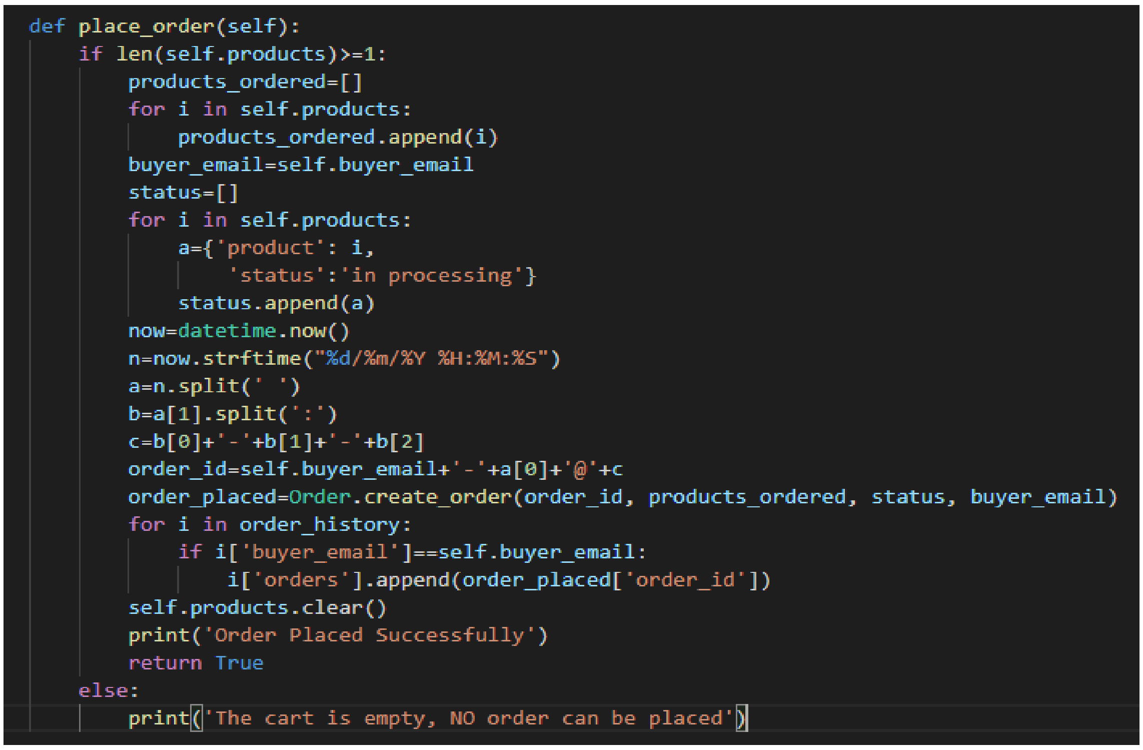
```python
order_placed=Order.create_order(order_id, products_ordered, status, buyer_email)
for i in order_history:
    if i['buyer_email']==self.buyer_email:
        i['orders'].append(order_placed['order_id'])
self.products.clear()
```

Fig 10 – Order history should include all the orders placed by the buyer.

When an order is placed, the order id is added in the orders attribute of Order History class. Thus, ensuring that the order history includes all the orders placed by that user.

11. Cart should have at least 1 item before placing order and cart should be empty after placing the order.

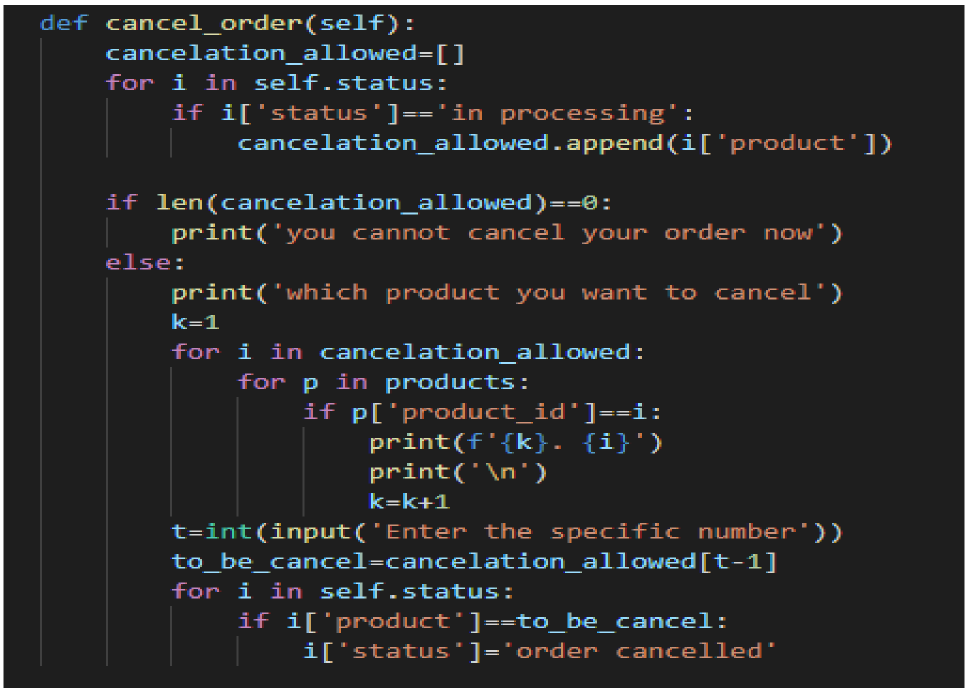
Before placing an order, the size of products array of class Cart is checked. To place an order the size must be greater than or equal to one or we can say that there should be at least one product in the cart. Also, when an order is placed, all the products in the cart are removed by removing all products from the array.



```python
def place_order(self):
    if len(self.products)>=1:
        products_ordered=[]
        for i in self.products:
            products_ordered.append(i)
        buyer_email=self.buyer_email
        status=[]
        for i in self.products:
            a={'product': i,
               'status':'in processing'}
            status.append(a)
        now=datetime.now()
        n=now.strftime("%d/%m/%Y %H:%M:%S")
        a=n.split(' ')
        b=a[1].split(':')
        c=b[0]+'-'+b[1]+'-'+b[2]
        order_id=self.buyer_email+'-'+a[0]+'@'+c
        order_placed=Order.create_order(order_id, products_ordered, status, buyer_email)
        for i in order_history:
            if i['buyer_email']==self.buyer_email:
                i['orders'].append(order_placed['order_id'])
        self.products.clear()
        print('Order Placed Successfully')
        return True
    else:
        print('The cart is empty, NO order can be placed')
```

Fig 11 – Cart should have at least one item for placing an order and after placing the order of the item, cart should be empty.

12. Order can be cancelled if its status is “in processing”.



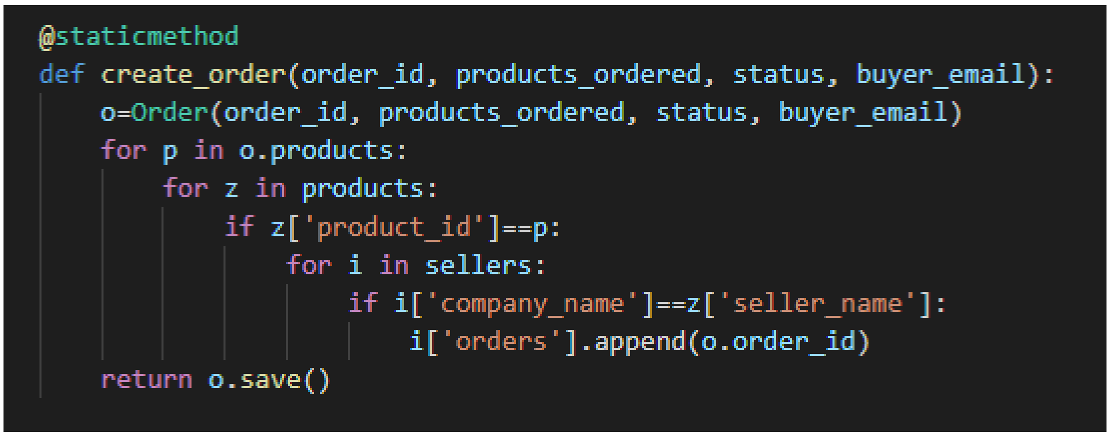
```python
def cancel_order(self):
    cancelation_allowed=[]
    for i in self.status:
        if i['status']=='in processing':
            cancelation_allowed.append(i['product'])

    if len(cancelation_allowed)==0:
        print('you cannot cancel your order now')
    else:
        print('which product you want to cancel')
        k=1
        for i in cancelation_allowed:
            for p in products:
                if p['product_id']==i:
                    print(f'{k}. {i}')
                    print('\n')
                    k=k+1
        t=int(input('Enter the specific number'))
        to_be_cancel=cancelation_allowed[t-1]
        for i in self.status:
            if i['product']==to_be_cancel:
                i['status']='order cancelled'
```

Fig 12 – Order can be cancelled only if the status is “IN PROCESSING”

Before cancelling an order for a specific product, first the status is checked. If it is still in processing that is the order is not shipped yet then the buyer can cancel the order for that product.

13. Seller can update the status of only those orders which includes the products offered by that seller.

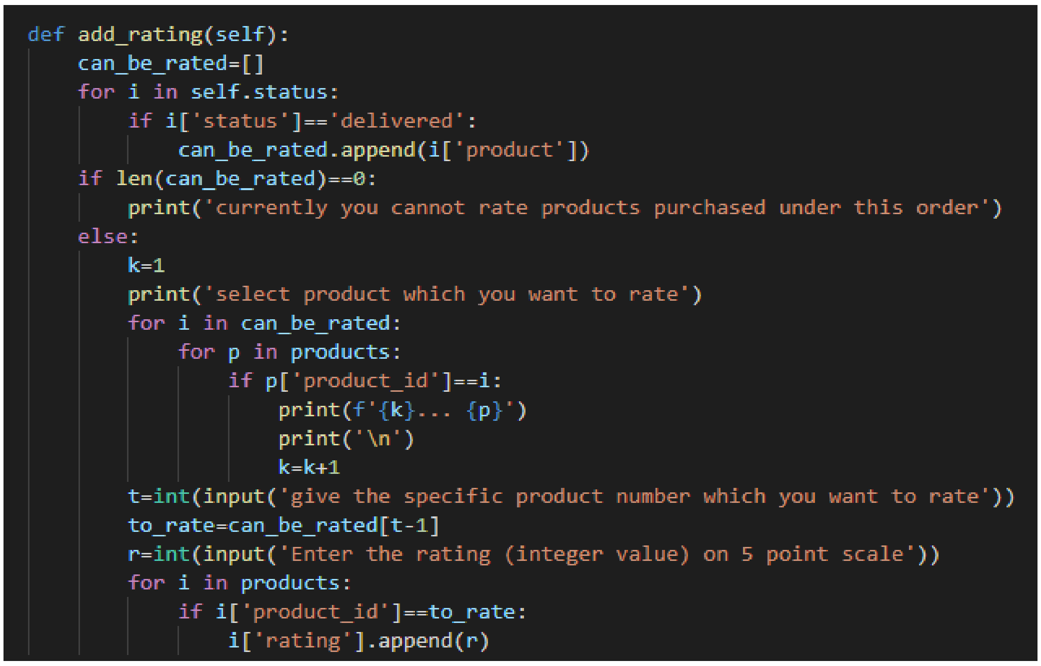


```python
@staticmethod
def create_order(order_id, products_ordered, status, buyer_email):
    o=Order(order_id, products_ordered, status, buyer_email)
    for p in o.products:
        for z in products:
            if z['product_id']==p:
                for i in sellers:
                    if i['company_name']==z['seller_name']:
                        i['orders'].append(o.order_id)
    return o.save()
```

Fig 3 – Updation of the status of the products by seller.

When an order is created/placed, then the sellers for all the products included in the order get the order id linked with them. This is done by adding the order id in the orders array of Seller class. When the seller wants to update the status of the order then he can do so by selecting the order id from the array. Thus, the seller can only update the status of orders which has the products offered by that seller.

14. Product can be rated only after it is delivered.



```python
def add_rating(self):
    can_be_rated=[]
    for i in self.status:
        if i['status']=='delivered':
            can_be_rated.append(i['product'])
    if len(can_be_rated)==0:
        print('currently you cannot rate products purchased under this order')
    else:
        k=1
        print('select product which you want to rate')
        for i in can_be_rated:
            for p in products:
                if p['product_id']==i:
                    print(f'{k}... {p}')
                    print('\n')
                    k=k+1
        t=int(input('give the specific product number which you want to rate'))
        to_rate=can_be_rated[t-1]
        r=int(input('Enter the rating (integer value) on 5 point scale'))
        for i in products:
            if i['product_id']==to_rate:
                i['rating'].append(r)
```

Fig 14 – Rating of products can be done only after the product is delivered.

When a buyer wants to rate a product ordered under a order id, the system will first check the status whether the product is delivered or not. If it is delivered only then the buyer can add the rating.