## Paul Bosonetto

## Jacob Ballew

## Peter Thompson

## Dahquan Williams

## **Specification Description for WebMarket**

This system implements logic and functionality that allows a client/customer to purchase products through an online service called Web Market. The customer can place orders, reorder previously purchased products and return products. The main logic of this subsystem is the WebMarket class which handles and encapsulates the details of ordering, recording and returning products. The customer does not know the details of what is going on “behind the curtains.” They merely place an order and WebMarket handles the rest. This subsystem draws upon the Command Design Pattern.

**Goals:**

-Easy for the Customer and straightforward for the vendor.

-Of benefit to the Customer – find them a lower price for a particular product than they are able to find themselves (at least, than they are able to find without significant time and effort).

-Maintain records of all transactions. This is for a variety of reasons – most importantly for reordering capabilities, but also for possible future expansions into areas like product recommendation.

-The system should be robust and responsive to error. It should not fail upon error, but rather try to catch the error and handle it.

**Buying Products**

Using a front-end, the customer places an order for a product. If the customer places a new order, they pass the name of the product onto the WebMarket. The WebMarket then queries a list of participating vendors and requests their price for said product. WebMarket can then perform a comparison of prices to find the lowest price. Once the lowest price has been found, WebMarket places the order with the corresponding vendor. The WebMarket handles the transference of payment from buyer to seller and then logs the transaction details. Logging the transaction aides in the reorder process since the client has been a customer before.

**Reordering previously purchased products**

If the customer has placed an order for a particular product before, they can reorder it. The customer class checks its own history to see if that particular customer is eligible for a reorder. If not, they are asked if they’d like to place a new order for the product.If a customer is reordering, the WebMarket will skip the step of finding the lowest price and will instead reorder from the same vendor who fulfilled the original order. The idea here is to save the customer time if they already know exactly what they want.

**Returning Previously purchased products**

If a customer places a return order, the Customer class will check the desired return product against its order history to make sure they are eligible for a return (have actually bought the product). If so, then the customer may place a return product request by passing the name of the product to the WebMarket. Using the details stored from the original buy order, WebMarket alerts the appropriate vendor of the return request. The vendor then surrenders the money to WebMarket who can then provide the customer with their refund amount. At this point the customer can now send back the item to WebMarket who will deliver the item to the vendor from which it came.

**Use Case Text Descriptions for WebMarket**

|  |  |
| --- | --- |
| *Use case name* | BuyProduct |
| *Participating actors* | Customer, WebMarket, Vendor |
| *Flow of events* | 1. Customer places an order with the WebMarket using the name of a product 2. WebMarket queries the list of previous transactions    1. WebMarket finds no previous transaction involving this Customer and this product 3. WebMarket queries all Vendors the price of the product if the Vendor has it in stock 4. WebMarket places the order with the Vendor that has the lowest price for the product 5. WebMarket takes payment from Customer 6. WebMarket gives payment to Vendor 7. Vendor allows WebMarket to access the product 8. WebMarket delivers product to Customer 9. WebMarket logs the details of the order with Customer, Vendor, product and price information |
| *Entry condition* | * Customer has an interest in buying a product * A Vendor has that product in stock |
| *Exit conditions* | * Customer now owns the product   + Customer surrenders money * Vendor has one fewer product in stock   + Vendor gains money * WebMarket has the details of the transaction logged as a previous order |

|  |  |
| --- | --- |
| *Use case name* | BuyAgain |
| *Participating actors* | Customer, WebMarket, Vendor |
| *Flow of events* | 1. Customer places an order with the WebMarket using the name of a product 2. WebMarket queries the list of previous orders    1. WebMarket finds a previous order involving this Customer and this product 3. WebMarket queries all Vendors the price of the product if the Vendor has it in stock    1. WebMarket applies a discount (5%) to the product price 4. WebMarket places the order with the Vendor that has the lowest price for the product 5. WebMarket takes payment from Customer 6. WebMarket gives payment to Vendor 7. Vendor allows WebMarket to access the product 8. WebMarket delivers product to Customer 9. WebMarket logs the details of the order with Customer, Vendor, product and price information |
| *Entry condition* | * Customer has an interest in buying a product   + Customer has purchased this product in the past * A Vendor has that product in stock * WebMarket has a log of the transaction in which this Customer has purchased this product previously |
| *Exit conditions* | * Customer now owns the product   + Customer surrenders money * Vendor has one fewer product in stock   + Vendor gains money * WebMarket has the details of the transaction logged as a previous order |

|  |  |
| --- | --- |
| *Use case name* | ReturnProduct |
| *Participating actors* | Customer, WebMarket, Vendor |
| *Flow of events* | 1. Customer places a return order with the WebMarket using the name of a product 2. WebMarket queries the list of previous orders    1. WebMarket finds a previous order involving this Customer and this product    2. WebMarket identifies the Vendor from which this product was originally purchased 3. WebMarket alerts Vendor of the return    1. Vendor surrenders money to WebMarket    2. WebMarket collects the money from Vendor 4. WebMarket collects product from Customer    1. WebMarket distributes money to Customer in exchange 5. WebMarket delivers used product to the Vendor 6. WebMarket changes the original purchase order to reflect the return |
| *Entry condition* | * Customer has a product that was purchased by a Vendor via the WebMarket   + Customer would like to return the product * WebMarket has a record of that order |
| *Exit conditions* | * Customer surrenders the product   + Customer gains money * Vendor surrenders money   + Vendor gains used product * WebMarket has changed the original purchase order to reflect that the item has been returned |

**WebMarket Service Design Description**

Our system is designed to model a WebMarket service similar to, but not exactly like, ebay or Amazon. A client, called Customer, uses a web service, called WebMarket in our implementation, to either buy, reorder, or return a product. The WebMarket service handles the order request by communicating with Vendors. The Customer is shielded from the many aspects that go into placing the order, and is required to only do the most basic of operations to buy, reorder, or return a product.

The **Order Class** describes an order, reorder, or return for a specific product. It also includes the identification of the Customer who created the order, as well as the Vendor who fulfills it, though the Vendor field will not be initialized until the WebMarket has selected the best Vendor to fulfill the Order. An order will also keep track of whether or not it has been fulfilled. Due to the comprehensive nature of Order objects, they are both used for actions (placing orders) and record keeping (order history).

The **Product Class** describes a product sold by a Vendor. It consists of a descriptive name and a price. This price may vary by Vendor.

The **Vendor Class** describes a vendor who sells products through the WebMarket. Each Vendor object contains a list of Products and their prices for each products. These prices may be different than other Vendors. Each Vendor object is identified by its name. The Vendor class has a method to return its best (cheapest) price for a product. The Vendor class also has private helper methods that facilitate the order process in conjunction with the WebMarket. A Vendor object will keep track of its order history.

The **Customer Class** describes a Customer using the WebMarket. It allows a client to order, reorder, or return a product. It includes a private helper method that collects a Customer’s payment with any order except a return. A Customer object will keep track of its order history.

The **WebMarket Class** is the most complex. It connects Customer and Vendor, facilitating and processing transactions between them. It iterates through a list of Vendors, queries each one, and selects the one with the lowest price to place a Customer’s order with. It coordinates payment between the Customer and Vendor in the case of a purchase, as well as delivery of the product. It processes a Customer’s return order by contacting the vendor and arranging for Customer return shipment and Vendor refund payment. It checks its global order history to ensure that a Customer is eligible for a reorder discount (i.e. they are indeed **re**ordering). The meat of this is done in a host of private helper methods behind the scenes, making the process easy and simple for both Vendor and Customer parties.

**Associations and Compositions:**

As shown in the Class Diagram, there will only be one WebMarket Object in any WebMarket system. It will associate with any number of Customer objects. It must associate with at least one Vendor object, but will likely associate with many. The system will not work if there are no Vendors from which to purchase products. The WebMarket will contain any amount of orders. Initially, it may contain zero orders, as the presence of a market does not guarantee an immediate purchase.

Each Vendor object will consist of one or more Product objects. This is the Vendor’s inventory, and a Vendor is not allowed in the market if it does not have any inventory to sell. Like a WebMarket, Vendor will consist of any amount of orders, including zero. Arrival of a Vendor to the market does not guarantee them a sale.

A Customer object will contain any amount of Order objects, this number being zero until they have placed their first order. They will only interact directly with a single WebMarket object.

**Design Decisions**

The most basic and fundamental decision we made in our design was to have a powerful and comprehensive WebMarket that could accomplish many different things while keeping the customer and vendor shielded from the complex details. From the customer’s end, anything they want to accomplish can be done by invoking a single method (which in turn invokes others). The vendor’s path is only slightly more complicated.

One of the most important choices we made was to have each customer keep track of their own order history, and to have the WebMarket keep track of all orders placed within the market. This addition greatly increased the possibility of what our system could do, allowing us to easily implement a reorder method and a return method. It also allows an administrator of the system to easily look through the history of the WebMarket in the case of an error. This design also opens future options for explorations into product recommendations and market analysis.

Embedded in this decision was the decision to make Orders their own object. This choice had several advantages for us. Rather than simply specifying order parameters and then discarding them after they were done, or then building them into some sort of formatted list, we created an object to store orders from the beginning. This made them easy to store and to pass through to other classes and methods. To store them, we simply created a list that held Order objects. And to pass them to other methods, we then had one object to send, rather than four or five parameters.

**Sequence of Events**

As shown in the Sequence Diagram, once things are initialized, the first event is started by a customer. The customer will decide to either place a new order, reorder a product, or return a product (the latter two only being available if previous orders exist). No matter which order a customer places, the next step is for that order to be passed onto the WebMarket. The WebMarket will process that order and take the appropriate actions, which differ for each order type.

In the case of a buy order, the customer will call the WebMarket finds the lowest price of a product by querying its list of vendors. It selects the vendor with the best price and places the order with them, rewarding vendors who are the most competitive. The WebMarket then transfers payment from the customer to the vendor, and the vendor delivers the product to the customer.

The second possible event is a reorder. The customer places a reorder in a similar way to a buy order, but since it calls the WebMarket’s *placeNewReorderOrder(Order order)* method instead of *placeNewOrder(Order order)* the WebMarket knows to process it differently. In this case the vendor will already be known – the original seller – and the WebMarket will skip the step of searching through vendors for the lowest price. The vendor fulfills the order again, payment is transferred from customer to vendor through the WebMarket, and the vendor delivers the product to the customer.

The third possible event is a return order. Calling *placeNewReturnOrder(Order order)* in WebMarket the customer distinguishes the order type. The vendor in this case is known, as the return order is fulfilled with the original vendor the product was purchased from. The WebMarket delivers the product from the customer back to the vendor and facilitates the refund from the vendor to the customer, completing the return.

**Testing the Components of the WebMarket**

**CustomerTest**

Setup: Instances of a Vendor, WebMarket and Customer objects are created. The Customer’s market is the WebMarket, and the Vendor is added to the WebMarket. A *goodProductName* is defined as “Frisbee” and occurs in the Vendor’s list of products. A *badproductName* is defined as “Ferzbee” and does not match any products in the Vendor’s list.

1. buyProduct(String productName) : void
   1. *productName* invalid : throws *ProductNotFoundException*
      1. Customer tries to buy a product using a product name that is not found: “Ferzbee”.
      2. *ProductNotFoundException* is thrown.
   2. *productName* valid : expect nothing
      1. Customer tries to buy a product with a valid product name: “Frisbee”.
      2. No unusual activity occurs.
2. getMarket() : WebMarket
   1. Assert the correct WebMarket is returned
      1. Customer’s *getMarket()* method is called.
      2. The returned WebMarket object matches the WebMarket object we expect.
3. reorderProduct(String productName) : void
   1. *productName* invalid : throws *ProductNotFoundException*
      1. Customer tries to reorder a product using a product name that is not valid: “Ferzbee”
      2. *ProductNotFoundException* is thrown.
   2. *productName* found (previously ordered) : expect no unusual activity
      1. Customer tries to reorder a product using a product name that is found in that Customer’s recent orders: “Frisbee”.
      2. No unusual activity occurs.
   3. *productName* not found (not previously ordered) : expect no unusual activity
      1. Customer tries to reorder a product name that is not found in that Customer’s recent orders: “Yo-Yo”.
4. returnProduct(String productName) : void
   1. *productName* not found : throws *ProductNotFoundException*
      1. Customer tries to return a product using a product name that is not valid: “Ferzbee”.
      2. *ProductNotFoundException* is thrown
   2. *productName* not found (not previously ordered) : throws *ProductNotFoundException*
      1. Customer tries to return a product using a product name that is valid, but has never been ordered: “Frisbee”.
      2. *ProductNotFoundException* is thrown
   3. *productName* found (previously ordered) : *orderType* is “return”
      1. Customer purchases “Frisbee”.
      2. Customer tries to return “Frisbee”.
      3. The order type is retrieved using the Customer’s *getOrderByProductName()* method
      4. The returned value is equal to “return”, signifying a return order.
5. setMarket(WebMarket market) : void
   1. Set the Customer’s *market* attribute to refer to a different WebMarket object
      1. Create a new WebMarket
      2. Use the Customer’s *setMarket()* to assign the new WebMarket
      3. Use the Customer’s *getMarket()* to retrieve it’s *market* attribute
      4. Assert that the WebMarket returned equals the WebMarket expected

**VendorTest**

Setup: A Vendor object is created. A *goodProductName* is defined as “Frisbee” and occurs in the Vendor’s list of products. A *badproductName* is defined as “Ferzbee” and does not match any products in the Vendor’s list.

1. getPriceOfProduct(String productName) : double
   1. *productName* found : assert the correct price of the product is returned
      1. The testing method *setProductPrice(goodProductName, 2.5)* sets the price of a “Frisbee” to $2.50.
      2. The Vendor’s *getPriceOfProduct()* method returns a double
      3. The returned value matched the expected value of 2.5.
   2. *productName* not found : throws *ProductNotFoundException*
      1. The Vendor’s *getPriceOfProduct()* method is passed the *badProductName*
      2. *ProductNotFoundException* is thrown.
2. processRefund(Order order) : void
   1. Expect no unusual activity
      1. The Vendor’s *processRefund()* is passed an Order
      2. No unusual activity occurs.

**WebMarketTest**

Setup:

1. addVendor(Vendor vendor) : void
   1. Vendor is added successfully. No unusual activity occurs.
2. placeNewOrder(Order order) : void
   1. Order type is “buy” : Order status should be “complete” and type “buy”
      1. Vendor is added to the WebMarket
      2. A new Order is created with the type “buy”
      3. The order is placed with the WebMarket
      4. The order status is retrieved by the order’s *getOrderStatus*() method
      5. The order type is retrieved by the order’s *getOrderType()* method
      6. Assert the returned order status equals “complete”
      7. Assert the returned order type equals “new”
   2. Order type is “reorder” : Order status should be “complete” and type “reorder”
      1. Vendor is added to the WebMarket
      2. A new Order is created with the type “reorder” and a Vendor specified.
      3. The order is placed with the WebMarket
      4. The order status is retrieved by the order’s *getOrderStatus*() method
      5. The order type is retrieved by the order’s *getOrderType()* method
      6. Assert the returned order status equals “complete”
      7. Assert the returned order type equals “reorder”
   3. Order type is “return” : Order status should be “complete” and type “return”
      1. Vendor is added to the WebMarket
      2. A new Order is created with the type “return” and a Vendor specified.
      3. The order is placed with the WebMarket
      4. The order status is retrieved by the order’s *getOrderStatus*() method
      5. The order type is retrieved by the order’s *getOrderType()* method
      6. Assert the returned order status equals “complete”
      7. Assert the returned order type equals “return”
3. placeNewReorderOrder(Order order) : void
   1. Order is valid : Order status should be “complete” and type “reorder”
      1. Vendor is added to the WebMarket
      2. A new Order is created with the type “reorder” and a Vendor specified.
      3. The order is placed with the WebMarket
      4. The order status is retrieved by the order’s *getOrderStatus*() method
      5. The order type is retrieved by the order’s *getOrderType()* method
      6. Assert the returned order status equals “complete”
      7. Assert the returned order type equals “reorder”
4. placeNewReturnOrder(Order order) : void
   1. Order is valid : Order status should be “complete”
      1. Vendor is added to the WebMarket
      2. A new Order is created with the type “return” and a Vendor specified.
      3. The order is placed with the WebMarket
      4. The order status is retrieved by the order’s *getOrderStatus*() method
      5. The order type is retrieved by the order’s *getOrderType()* method
      6. Assert the returned order status equals “complete”
      7. Assert the returned order type equals “return”







