



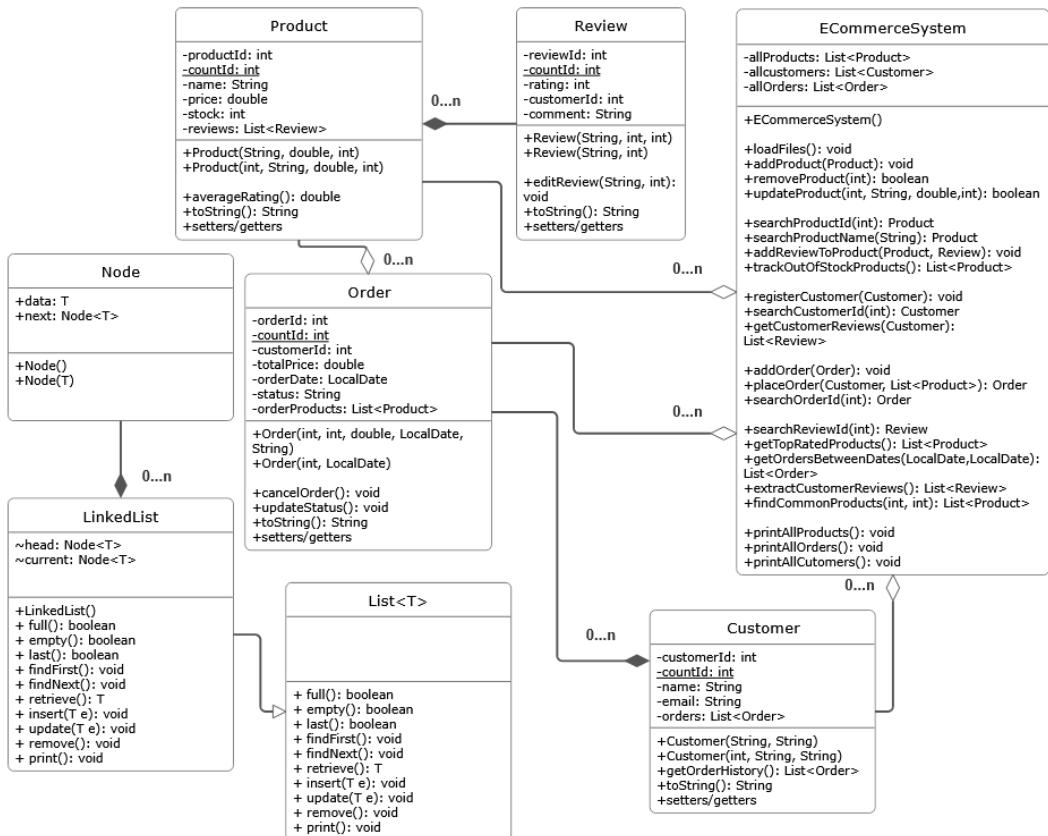
CSC212 Project
Data Structures
E-Commerce Inventory &
Order Management System
PHASE 1

Yazan Almuzaini 445103365

Abdulaziz Alnahedh 445102328

Sultan Almandeel 445100971

Class Diagram:



All Required Methods, with Big O analysis:

Only a single data structure type was needed, which is the linked list.

Method: addProduct

```
● ● ●  
1  public void addProduct(Product p) {  
2      try {  
3          BufferedWriter writer = new BufferedWriter(new  
4              FileWriter(PRODUCTS_FILE, true));  
5          writer.newLine();  
6          writer.write(p.getProductId() + "," + p.getName() +  
7              "," + p.getPrice() + "," + p.getStock());  
8          writer.close();  
9          allProducts.insert(p);  
10     } catch (IOException e) {  
11         System.err.println(e.getMessage());  
12     }  
13 }
```

Description: adds a new product and appends its information to the CSV file.

- Time complexity: O(1)
- Space Complexity: O(1)

Method: removeProduct

```
1 public boolean removeProduct(int productId) {  
2     Product p = searchProductId(productId);  
3     if (p != null) {  
4         p.setStock(0);  
5         return true;  
6     }  
7     return false;  
8 }  
9
```

Description: Removes a product by its ID by searching for it and setting its stock to zero if found.

- Time complexity: O(n)
- Space Complexity: O(1)

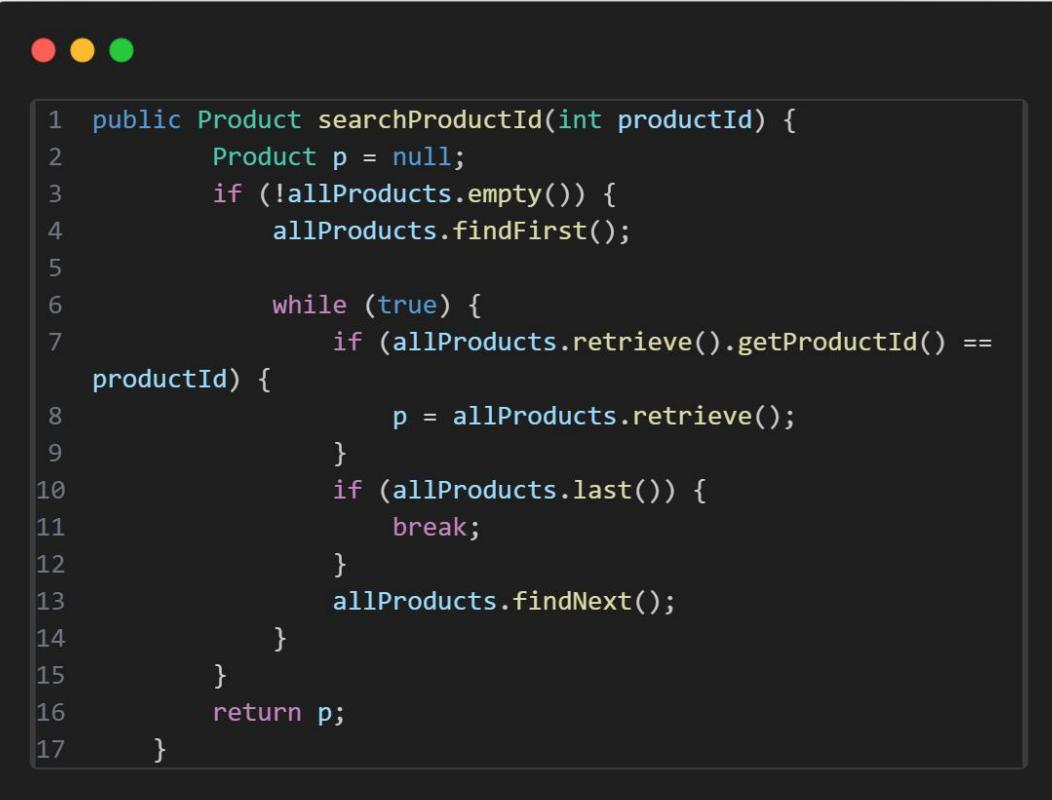
Method: updateProduct

```
1 public boolean updateProduct(int id, String name, double price
, int stock) {
2     Product p = searchProductId(id);
3     if (p != null) {
4         p.setName(name);
5         p.setPrice(price);
6         p.setStock(stock);
7         return true;
8     }
9     return false;
10}
11
```

Description: Updates the details (name, price, and stock) of a product found by its ID.

- Time complexity: O(n)
- Space Complexity: O(1)

Method: searchProductId



The screenshot shows a Java code editor window with a dark theme. At the top left are three colored window control buttons: red, yellow, and green. The code itself is a single method named `searchProductId`. It starts with a public return type `Product`, followed by the method name `searchProductId` and a parameter `int productId`. Inside the method, it initializes a `Product` object `p` to `null`. It then checks if the list `allProducts` is empty. If not, it calls `allProducts.findFirst()`. A `while` loop then iterates as long as `true`. Inside the loop, it checks if the current product's ID matches the `productId`. If it does, it sets `p` to the retrieved product and exits the loop. If the end of the list is reached (`allProducts.last()`), it breaks out of the loop. After exiting the loop, it calls `allProducts.findNext()` to move to the next item in the list. Finally, it returns the found product `p`.

```
1 public Product searchProductId(int productId) {
2     Product p = null;
3     if (!allProducts.empty()) {
4         allProducts.findFirst();
5
6         while (true) {
7             if (allProducts.retrieve().getProductId() ==
8                 productId) {
9                 p = allProducts.retrieve();
10            }
11            if (allProducts.last()) {
12                break;
13            }
14            allProducts.findNext();
15        }
16    }
17    return p;
18 }
```

Description: Searches the product list for a product with the specified ID and returns it if found, otherwise, returns null.

- Time complexity: $O(n)$
- Space Complexity: $O(1)$

Method: searchProductName

```
1 public Product searchProductName(String name) {  
2     Product p = null;  
3     if (!allProducts.empty()) {  
4         allProducts.findFirst();  
5  
6         while (true) {  
7             if (allProducts.retrieve().getName().  
8                 equalsIgnoreCase(name)) {  
9                 p = allProducts.retrieve();  
10            }  
11            if (allProducts.last()) {  
12                break;  
13            }  
14            allProducts.findNext();  
15        }  
16    }  
17    return p;  
18}
```

Description: Searches the product list for a product whose name matches the given name and returns it if found, otherwise, returns null.

- Time complexity: O(n)
- Space Complexity: O(1)

Method: trackOutOfStockProducts

```
1 public List<Product> trackOutOfStockProducts() {  
2     List<Product> tosp = new LinkedList<>();  
3     if (!allProducts.empty()) {  
4         allProducts.findFirst();  
5  
6         while (true) {  
7             if (allProducts.retrieve().getStock() == 0) {  
8                 tosp.insert(allProducts.retrieve());  
9             }  
10            if (allProducts.last()) {  
11                break;  
12            }  
13            allProducts.findNext();  
14        }  
15    }  
16    return tosp;  
17}  
18}  
19
```

Description: Creates and returns a list of all products that currently have zero stock.

- Time complexity: $O(n)$
- Space Complexity: $O(m)$

Method: registerCustomer



A screenshot of a Java code editor window. The title bar shows three colored window control buttons (red, yellow, green). The main area contains the following Java code:

```
1 public void registerCustomer(Customer c) {
2     try {
3         BufferedWriter writer = new BufferedWriter(new
4             FileWriter(CUSTOMERS_FILE, true));
5         writer.newLine();
6         writer.write(c.getCustomerId() + "," + c.getName()
7             + "," + c.getEmail());
8         writer.close();
9         allCustomers.insert(c);
10    } catch (IOException e) {
11        System.err.println(e.getMessage());
12    }
13}
```

Description: Registers a new customer by writing their details to the customers file and adding them to the customer list.

- Time complexity: O(1)
- Space Complexity: O(1)

Method: searchCustomerId

```
1  public Customer searchCustomerId(int customerId) {  
2      Customer c = null;  
3      if (!allCustomers.empty()) {  
4          allCustomers.findFirst();  
5  
6          while (true) {  
7              if (allCustomers.retrieve().getCustomerId()  
== customerId) {  
8                  c = allCustomers.retrieve();  
9              }  
10             if (allCustomers.last()) {  
11                 break;  
12             }  
13             allCustomers.findNext();  
14         }  
15     }  
16     return c;  
17 }  
18 }
```

Description: Finds and returns the customer whose ID matches the given customer ID, returns null if no match is found.

- Time complexity: O(n)
- Space Complexity: O(1)

Method: addOrder

```
1 public void addOrder(Order o) {
2     StringBuilder orderPID = new StringBuilder();
3     orderPID.append('\'');
4     if (!o.getOrderProducts().empty()) {
5         o.getOrderProducts().findFirst();
6         while (true) {
7             if (orderPID.length() > 1) {
8                 orderPID.append(';');
9             }
10            orderPID.append(o.getOrderProducts().retrieve
11                ().getProductId());
12            if (o.getOrderProducts().last()) {
13                break;
14            }
15            o.getOrderProducts().findNext();
16        }
17    }
18    try {
19        BufferedWriter writer = new BufferedWriter(new
20            FileWriter(ORDERS_FILE, true));
21        writer.newLine();
22        writer.write(o.getOrderId() + "," + o.
23            getCustomerId() + "," + orderPID.toString() + "," + o.
24            getTotalPrice() + "," + o.getOrderDate() + "," + o.getStatus
25            ());
26    }
27 }
28 }
```

Description: adds a new order by writing its details (including product IDs, total price, date, and status) to the orders file and adding it to the order list.

- Time complexity: O(n)
- Space Complexity: O(m)

Method: placeOrder

```
1 public Order placeOrder(Customer c, List<Product> products) {  
2     Order order = new Order(c.getCustomerId(), LocalDate.  
now());  
3     double totalPrice = 0;  
4     if (!products.empty()) {  
5         products.findFirst();  
6         while (true) {  
7             Product p = products.retrieve();  
8             order.getOrderProducts().insert(p);  
9             totalPrice += p.getPrice();  
10            p.setStock(p.getStock() - 1);  
11            if (products.last()) {  
12                break;  
13            }  
14            products.findNext();  
15        }  
16    }  
17    order.setTotalPrice(totalPrice);  
18    c.getOrderHistory().insert(order);  
19    addOrder(order); // appends to csv file  
20    return order;  
21 }
```

Description: Creates a new order for a customer by adding the selected products, updating their stock numbers, calculating the total price, saving the order, and returning it.

- Time complexity: O(n)
- Space Complexity: O(1)

Method: searchOrderId

```
1  public Order searchOrderId(int orderId) {  
2      Order o = null;  
3      if (!allOrders.empty()) {  
4          allOrders.findFirst();  
5          while (true) {  
6              if (allOrders.retrieve().getOrderId() ==  
7                  orderId) {  
8                  o = allOrders.retrieve();  
9              }  
10             if (allOrders.last()) {  
11                 break;  
12             }  
13             allOrders.findNext();  
14         }  
15     }  
16     return o;  
}
```

Description: Searches the order list for an order with the specified ID and returns it if found, otherwise, returns null.

- Time complexity: $O(n)$
- Space Complexity: $O(1)$

Method: searchReviewId

```
1 public Review searchReviewId(int reviewId) {  
2     Review r = null;  
3     if (!allProducts.empty()) {  
4         allProducts.findFirst();  
5         while (true) {  
6             Product p = allProducts.retrieve();  
7             List<Review> reviews = p.getReviews();  
8  
9             if (!reviews.empty()) {  
10                 reviews.findFirst();  
11                 while (true) {  
12                     if (reviews.retrieve().getReviewId()  
13 == reviewId) {  
14                         r = reviews.retrieve();  
15                     }  
16                     if (reviews.last()) {  
17                         break;  
18                     }  
19                     reviews.findNext();  
20                 }  
21             }  
22  
23             if (allProducts.last()) {  
24                 break;  
25             }  
26             allProducts.findNext();  
27         }  
28     }  
29     return r;  
30 }
```

Description : Searches through all products and their reviews to find and return the review with the specified ID, returns null if not found..

- Time complexity: $O(p \cdot r)$ where p is the number of products and r is the number of reviews.
- Space Complexity: $O(1)$

Method: getTopRatedProducts

```
1 public List<Product> getTopRatedProducts() {  
2     //get 3 top rated products by average rating and return list  
3     List<Product> top = new LinkedList<>();  
4     if (allProducts.empty()) {  
5         return top;  
6     }  
7     Product first = null, second = null, third = null;  
8  
9     allProducts.findFirst();  
10    while (true) {  
11        Product p = allProducts.retrieve();  
12        double rating = p.averageRating();  
13  
14        if (first == null || rating > first.averageRating()  
15            ()) {  
16            third = second;  
17            second = first;  
18            first = p;  
19        } else if (second == null || rating > second.  
20            averageRating()) {  
21            third = second;  
22            second = p;  
23        } else if (third == null || rating > third.  
24            averageRating()) {  
25            third = p;  
26        }  
27  
28        if (allProducts.last()) {  
29            break;  
30        }  
31        allProducts.findNext();  
32    }  
33  
34    if (first != null) {  
35        top.insert(first);  
36    }  
37    if (second != null) {  
38        top.insert(second);  
39    }  
40    if (third != null) {  
41        top.insert(third);  
42    }  
43  
44    return top;  
45}
```

Description : Finds and returns the top three products with the highest average ratings by going through all products and comparing their ratings..

- Time complexity: $O(p \cdot r)$
- Space Complexity: $O(1)$

Method: getOrdersBetweenDates

```
1 public List<Order> getOrdersBetweenDates(LocalDate startDate
, LocalDate endDate) {
2     //get orders between two dates and return list
3     List<Order> result = new LinkedList<>();
4     if (!allOrders.empty()) {
5         Order o;
6         allOrders.findFirst();
7         while (true) {
8             o = allOrders.retrieve();
9             LocalDate date = o.getOrderDate();
10            if (date.isAfter(startDate) && date.isBefore(
endDate)) {
11                result.insert(o);
12            }
13            allOrders.findNext();
14            if (allOrders.last()) {
15                break;
16            }
17        }
18    }
19    return result;
20}
21}
```

Description: Retrieves and returns all orders whose dates are between the specified start and end dates.

- Time complexity: O(n)
- Space Complexity: O(m)

Method: getCustomerReviews

```
1 public List<Review> getCustomerReviews(Customer c) {
2     List<Review> reviewsByC = new LinkedList<>();
3     if (!allProducts.empty()) {
4         allProducts.findFirst();
5         while (true) {
6             List<Review> reviews = allProducts.retrieve
7                 ().getReviews();
8             if (!reviews.empty()) {
9                 reviews.findFirst();
10                while (true) {
11                    if (reviews.retrieve().getCustomerId
12                        () == c.getCustomerId()) {
13                        reviewsByC.insert(reviews.
14                            retrieve());
15                    }
16                    reviews.findNext();
17                    if (reviews.last()) {
18                        break;
19                    }
20                }
21                if (allProducts.last()) {
22                    break;
23                }
24                allProducts.findNext();
25            }
26        }
27    return reviewsByC;
28}
29}
```

Description: retrieves and returns all reviews written by a given customer by going through every product and its reviews.

- Time complexity: $O(p \cdot r)$
- Space Complexity: $O(m)$

Method: findCommonProducts

```
1 public List<Product> findCommonProducts(int customerId1, int
2   customerId2) {
3
4     //find common products reviewed by two customers with produc
5     t's average rating above 4.0(4.1, 4.2...5) and return list
6     List<Product> result = new LinkedList<>();
7     if (!allProducts.empty()) {
8       allProducts.findFirst();
9       while (true) {
10         Product p = allProducts.retrieve();
11         boolean reviewedBy1 = false;
12         boolean reviewedBy2 = false;
13         List<Review> revs = p.getReviews();
14         if (!revs.empty()) {
15           revs.findFirst();
16           while (true) {
17             Review r = revs.retrieve();
18             if (r.getCustomerId() == customerId1
19 ) {
20               reviewedBy1 = true;
21             }
22             if (r.getCustomerId() == customerId2
23 ) {
24               reviewedBy2 = true;
25             }
26           }
27
28           if (reviewedBy1 && reviewedBy2 && p.
29             averageRating() > 4.0) {
30               result.insert(p);
31             }
32             if (allProducts.last()) {
33               break;
34             }
35             allProducts.findNext();
36           }
37         return result;
38       }
39     }
```

Description: Finds and returns a list of products that were reviewed by both given customers and have an average rating greater than 4.0.

- Time complexity: $O(p \cdot r)$
- Space Complexity: $O(m)$