CSC212 Project

Data Structures

E-Commerce Inventory &

Order Management System

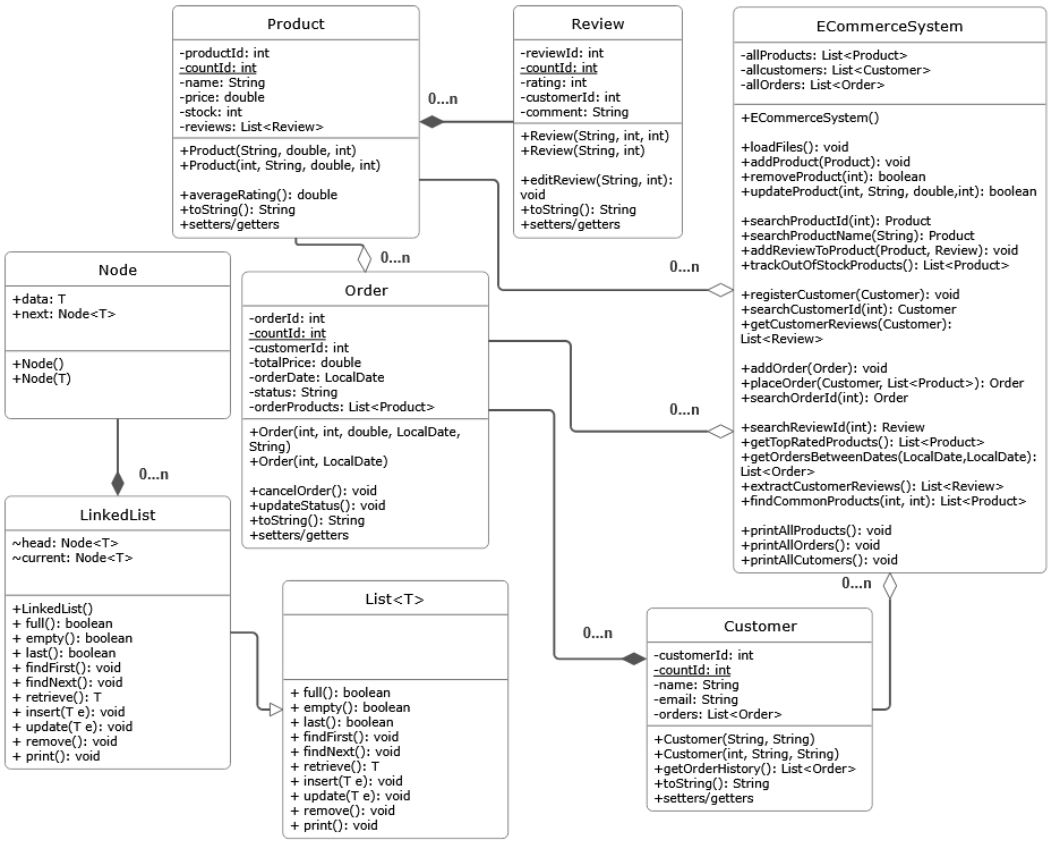
PHASE 1

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

```java
public void addProduct(Product p) {
        try {
            BufferedWriter writer = new BufferedWriter(new
FileWriter(PRODUCTS_FILE, true));
            writer.newLine();
            writer.write(p.getProductId() + "," + p.getName() +
 "," + p.getPrice() + "," + p.getStock());
            writer.close();
            allProducts.insert(p);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

```

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

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

Method: removeProduct

```java
public boolean removeProduct(int productId) {
        Product p = searchProductId(productId);
        if (p != null) {
            p.setStock(0);
            return true;
        }
        return false;
    }

```

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

```
public boolean updateProduct(int id, String name, double price
, int stock) {
        Product p = searchProductId(id);
        if (p != null) {
            p.setName(name);
            p.setPrice(price);
            p.setStock(stock);
            return true;
        }
        return false;
    }

```

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

```
public Product searchProductId(int productId) {
        Product p = null;
        if (!allProducts.empty()) {
            allProducts.findFirst();

            while (true) {
                if (allProducts.retrieve().getProductId() ==
productId) {
                    p = allProducts.retrieve();
                }
                if (allProducts.last()) {
                    break;
                }
                allProducts.findNext();
            }
        }
        return p;
    }
```

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

```
public Product searchProductName(String name) {
        Product p = null;
        if (!allProducts.empty()) {
            allProducts.findFirst();

            while (true) {
                if (allProducts.retrieve().getName().
equalsIgnoreCase(name)) {
                    p = allProducts.retrieve();
                }
                if (allProducts.last()) {
                    break;
                }
                allProducts.findNext();
            }

        }
        return p;
    }
```

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

```java
public List<Product> trackOutOfStockProducts() {
        List<Product> tosp = new LinkedList<>();
        if (!allProducts.empty()) {
            allProducts.findFirst();

            while (true) {
                if (allProducts.retrieve().getStock() == 0) {
                    tosp.insert(allProducts.retrieve());
                }
                if (allProducts.last()) {
                    break;
                }
                allProducts.findNext();
            }

        }
        return tosp;
    }

```

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

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

Method: registerCustomer

```
public void registerCustomer(Customer c) {
        try {
            BufferedWriter writer = new BufferedWriter(new
FileWriter(CUSTOMERS_FILE, true));
            writer.newLine();
            writer.write(c.getCustomerId() + "," + c.getName()
+ "," + c.getEmail());
            writer.close();
            allCustomers.insert(c);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

```

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)

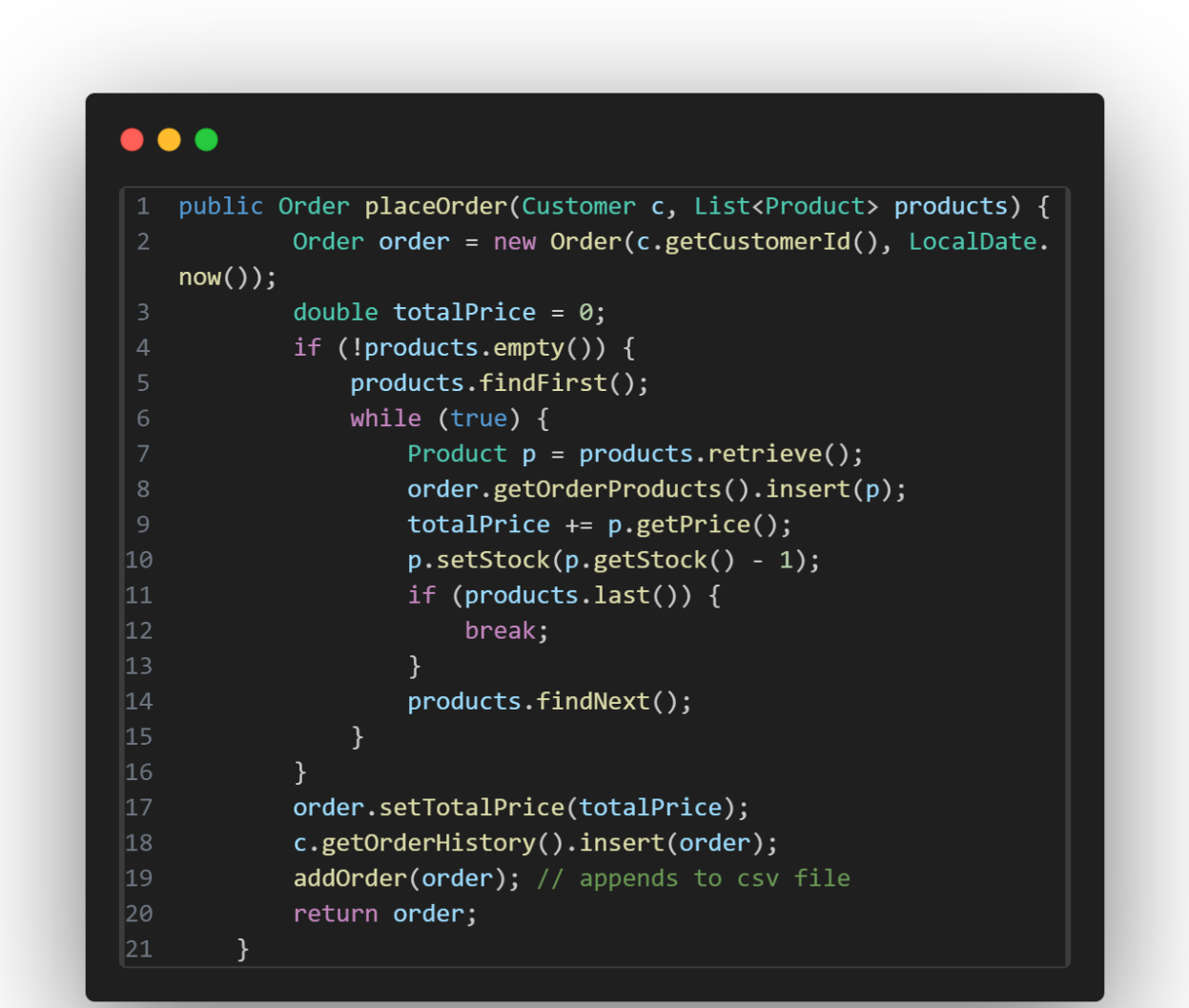
Method: addOrder

```
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
   ().getProductId());
11                 if (o.getOrderProducts().last()) {
12                     break;
13                 }
14                 o.getOrderProducts().findNext();
15             }
16             orderPID.append('"');
17         }
18         try {
19             BufferedWriter writer = new BufferedWriter(new
   FileWriter(ORDERS_FILE, true));
20             writer.newLine();
21             writer.write(o.getOrderId() + "," + o.
   getCustomerId() + "," + orderPID.toString() + "," + o.
   getTotalPrice() + "," + o.getOrderDate() + "," + o.getStatus
   ());
22             writer.close();
23             allOrders.insert(o);
24         } catch (IOException e) {
25             System.err.println(e.getMessage());
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



```java
public Order placeOrder(Customer c, List<Product> products) {
        Order order = new Order(c.getCustomerId(), LocalDate.
now());
        double totalPrice = 0;
        if (!products.empty()) {
            products.findFirst();
            while (true) {
                Product p = products.retrieve();
                order.getOrderProducts().insert(p);
                totalPrice += p.getPrice();
                p.setStock(p.getStock() - 1);
                if (products.last()) {
                    break;
                }
                products.findNext();
            }
        }
        order.setTotalPrice(totalPrice);
        c.getOrderHistory().insert(order);
        addOrder(order); // appends to csv file
        return order;
    }
```

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

```java
public Order searchOrderId(int orderId) {
        Order o = null;
        if (!allOrders.empty()) {
            allOrders.findFirst();
            while (true) {
                if (allOrders.retrieve().getOrderId() ==
orderId) {
                    o = allOrders.retrieve();
                }
                if (allOrders.last()) {
                    break;
                }
                allOrders.findNext();
            }
        }
        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

```java
public Review searchReviewId(int reviewId) {
        Review r = null;
        if (!allProducts.empty()) {
            allProducts.findFirst();
            while (true) {
                Product p = allProducts.retrieve();
                List<Review> reviews = p.getReviews();

                if (!reviews.empty()) {
                    reviews.findFirst();
                    while (true) {
                        if (reviews.retrieve().getReviewId()
== reviewId) {
                            r = reviews.retrieve();
                        }
                        if (reviews.last()) {
                            break;
                        }
                        reviews.findNext();
                    }

                }

                if (allProducts.last()) {
                        break;
                    }
                    allProducts.findNext();
            }
        }
        return r;
    }
```

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*r) where p is the number of products and r is the number of reviews.
- Space Complexity: O(1)

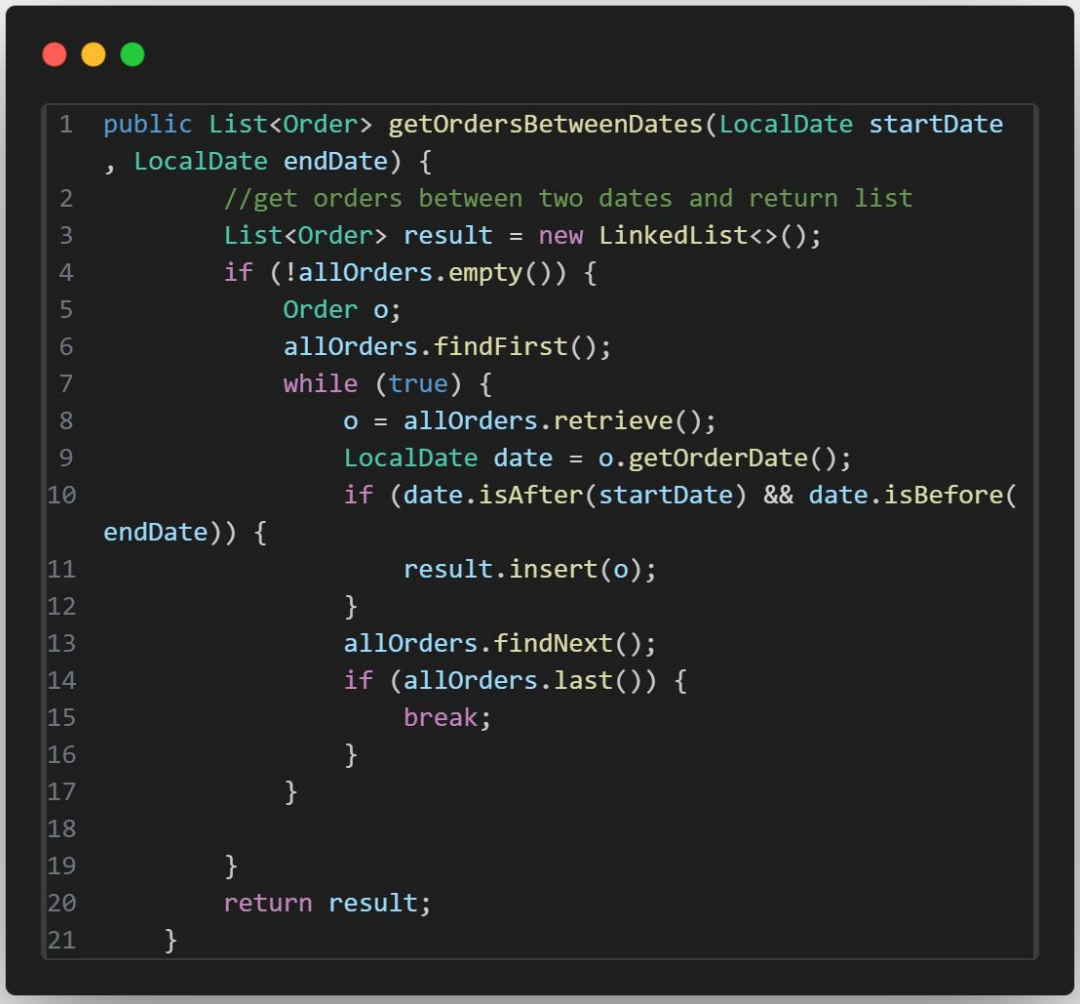
Method: getTopRatedProducts

```java
public List<Product> getTopRatedProducts() {

//get 3 top rated products by average rating and return list
        List<Product> top = new LinkedList<>();
        if (allProducts.empty()) {
            return top;
        }
        Product first = null, second = null, third = null;

        allProducts.findFirst();
        while (true) {
            Product p = allProducts.retrieve();
            double rating = p.averageRating();

            if (first == null || rating > first.averageRating
()) {
                third = second;
                second = first;
                first = p;
            } else if (second == null || rating > second.
averageRating()) {
                third = second;
                second = p;
            } else if (third == null || rating > third.
averageRating()) {
                third = p;
            }

            if (allProducts.last()) {
                break;
            }
            allProducts.findNext();
        }

        if (first != null) {
            top.insert(first);
        }
        if (second != null) {
            top.insert(second);
        }
        if (third != null) {
            top.insert(third);
        }

        return top;
    }
```

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*r)
- Space Complexity: O(1)

Method: getOrdersBetweenDates



```java
public List<Order> getOrdersBetweenDates(LocalDate startDate
, LocalDate endDate) {
        //get orders between two dates and return list
        List<Order> result = new LinkedList<>();
        if (!allOrders.empty()) {
            Order o;
            allOrders.findFirst();
            while (true) {
                o = allOrders.retrieve();
                LocalDate date = o.getOrderDate();
                if (date.isAfter(startDate) && date.isBefore(
endDate)) {
                    result.insert(o);
                }
                allOrders.findNext();
                if (allOrders.last()) {
                    break;
                }
            }

        }
        return result;
    }
```

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

```
public List<Review> getCustomerReviews(Customer c) {
        List<Review> reviewsByC = new LinkedList<>();
        if (!allProducts.empty()) {
            allProducts.findFirst();
            while (true) {
                List<Review> reviews = allProducts.retrieve
().getReviews();
                if (!reviews.empty()) {
                    reviews.findFirst();
                    while (true) {
                        if (reviews.retrieve().getCustomerId
() == c.getCustomerId()) {
                            reviewsByC.insert(reviews.
retrieve());
                        }

                        reviews.findNext();
                        if (reviews.last()) {
                            break;
                        }
                    }
                }
                if (allProducts.last()) {
                    break;
                }
                allProducts.findNext();

            }
        }
        return reviewsByC;

    }
```

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

- Time complexity: O(p*r)
- Space Complexity: O(m)

Method: findCommonProducts

```java
public List<Product> findCommonProducts(int customerId1, int customerId2) {

//find common products reviewed by two customers with product's average rating above 4.0(4.1, 4.2...5) and return list
        List<Product> result = new LinkedList<>();
        if (!allProducts.empty()) {
            allProducts.findFirst();
            while (true) {
                Product p = allProducts.retrieve();
                boolean reviewedBy1 = false;
                boolean reviewedBy2 = false;
                List<Review> revs = p.getReviews();
                if (!revs.empty()) {
                    revs.findFirst();
                    while (true) {
                        Review r = revs.retrieve();
                        if (r.getCustomerId() == customerId1) {
                            reviewedBy1 = true;
                        }
                        if (r.getCustomerId() == customerId2) {
                            reviewedBy2 = true;
                        }
                        if (revs.last()) {
                            break;
                        }
                        revs.findNext();
                    }
                }

                if (reviewedBy1 && reviewedBy2 && p.averageRating() > 4.0) {
                    result.insert(p);
                }
                if (allProducts.last()) {
                    break;
                }
                allProducts.findNext();
            }
        }
        return result;
    }

```

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*r)
- Space Complexity: O(m)