King Saud University
College of Computer and Information Sciences

CSC212 Course
1st Semester Fall 2025

E-Commerce Inventory & Order Management System

Phase # 1

Section #66510

NAME	ID
Amani Bin Oun	*445201317*
Ghala Almshegheh	*445202218*
Deema Alquwaei	*445200256*

Introduction

This project implements an E-Commerce Inventory & Order Management System designed to help businesses manage products, process customer orders, and analyze sales data. The system demonstrates practical applications of data structures in solving real-world business problems.

The primary goal is to create an efficient management platform that handles product inventory, customer information, order processing, and product reviews while providing analytical insights for business decision-making.

Classes And Methods

1. **Class Products**

   Attributes:
   - productId
   - name
   - price
   - stock
   - LinkedList <Integer> reviews = new LinkedList <Integer> ()

   Constructors:
   - Products()
   - Products(int productId, String name, double price, int stock)
   - Products(String fileName)
   -

   Methods:
   - getProductId()
   - setProductId(int productId)
   - getProductName()
   - setProductName(String name)
   - getProductPrice()
   - setProductPrice(double price)
   - getStock()
   - setStock(int stock)
   - addStock (int stock)
   - removeStock (int stock)
   - addReview(Integer Rating)
   - removeReview(Integer Rating)
   - getReviews ()
   - toString()
   - addProduct()
   - searchProducID()
   - removeProduct()
   - updateProduct()
   - searchProducName()
   - OutStockProducts()
   - checkProductID(int productId)
   - getProductData(int ProductId)

2. **Class Customers**

Attributes:
- customerId
- name
- email
- stock
- LinkedList <Integer> reviews = new LinkedList <Integer> ()

Constructors:
- Customers()
- Customers(int customerId, String name, String email)
- Customers(String fileName)
-

Methods:
- getCustomerId()
- setCustomerId(int customerId)
- getCusName()
- setCusName(String name)
- getCusEmail()
- setCusEmail(String email)
- getOrders()
- addOrder(Integer orderId)
- removeOrder(Integer target)
- toString()
- registerCustomer()
- placeOrderForCustomer(LinkedList<Products> products, LinkedList<Orders> orders)
- showOrderHistory()
- customerExists(int id)
- findCustomerById()
- printAllCustomers()

3. **Class Orders**

Attributes:

- orderId
- customerRef
- LinkedList<Integer> products = new LinkedList <Integer> ()
- totalPrice
- orderDate
- status

Constructors:

- Orders()
- Orders(int orderId, int customerRef, Integer[] ProductIDs, double totalPrice, String orderDate, String status)
- Orders(String fileName)

Methods:

- getOrderId()
- setOrderId(int orderId)
- getCustomerRef()
- setCustomerRef(int customerRef)
- getProducts()
- setProducts(LinkedList<Integer> products)
- getTotalPrice()
- setTotalPrice(double totalPrice)
- getOrderDate()
- setOrderDate(String orderDate)
- getStatus()
- setStatus(String status)
- addProduct(Integer product)
- removeProduct(Integer P)
- toString()
- createOrder(int customerId, LinkedList<Products> productsList)
- checkProductExists(int pid, LinkedList<Products> productsList)
- cancelOrder(int id)
- UpdateOrder(int orderID)
- searchById(int id)
- getOrdersBetween(String start, String end)
- checkOrderID(int oid)

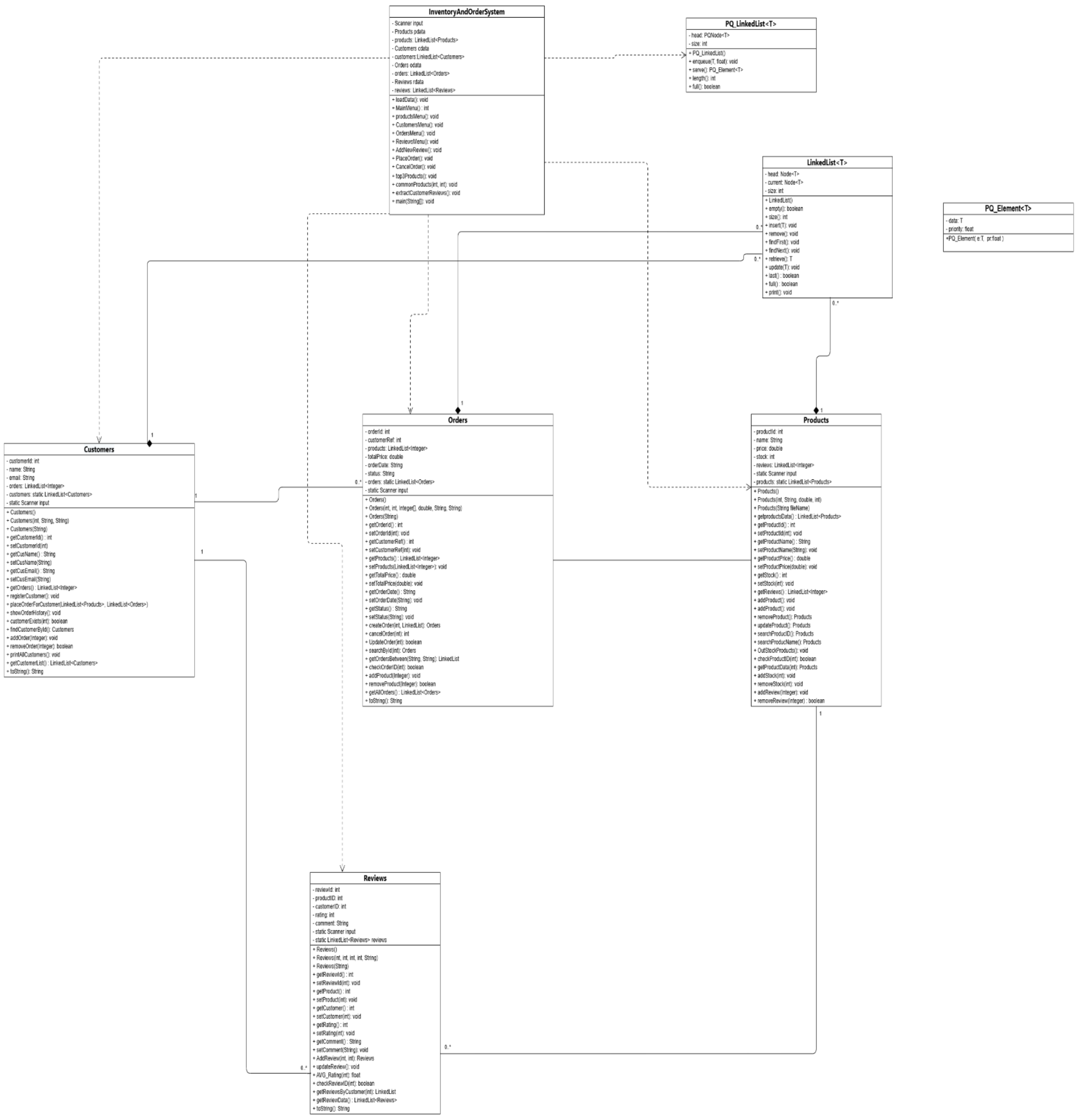
4. **Class Reviews**

Attributes:
- reviewId
- productID
- customerID
- rating
- comment

Constructors:
- Reviews()
- Reviews(int reviewId, int productID, int customerID, int rating, String comment)
- Reviews(String fileName)

Methods:
- getReviewId()
- setReviewId(int reviewId)
- getProduct()
- setProduct(int product)
- getCustomer()
- setCustomer(int customer)
- getRating()
- setRating(int rating)
- getComment()
- setComment(String comment)
- toString()
- AddReview(int customerID, int productID)
- updateReview()
- checkReviewID(int rID)
- getReviewsByCustomer(int customerId)
- AVG_Rating(int pid)

Class Diagram


InventoryAndOrderSystem
- Scanner input
- Products pdata
- products: LinkedList<Products>
- Customers cdata
- customers:LinkedList<Customers>
- Orders odata
- orders: LinkedList<Orders>
- Reviews rdata
- reviews: LinkedList<Reviews>
+ loadData(): void
+ MainMenu() : int
+ productsMenu(): void
+ CustomersMenu(): void
+ OrdersMenu(): void
+ ReviewsMenu(): void
+ AddNewReview(): void
+ PlaceOrder(): void
+ CancelOrder(): void
+ top3Products(): void
+ commonProducts(int, int): void
+ extractCustomerReviews(): void
+ main(String[]): void
PQ_LinkedList<T>
- head: PQNode<T>
- size: int
+ PQ_LinkedList()
+ enqueue(T, float): void
+ serve(): PQ_Element<T>
+ length(): int
+ full(): boolean
LinkedList<T>
- head: Node<T>
- current: Node<T>
- size: int
+ LinkedList()
+ empty(): boolean
+ size(): int
+ insert(T): void
+ remove(): void
+ findFirst(): void
+ findNext(): void
+ retrieve(): T
+ update(T): void
+ last() : boolean
+ full() : boolean
+ print(): void
PQ_Element<T>
- data: T
- priority: float
+PQ_Element(e:T, pr:float)
0..*
0..*
0..*
1
1
1
Customers
- customerId: int
- name: String
- email: String
- orders: LinkedList<Integer>
- customers: static LinkedList<Customers>
- static Scanner input
+ Customers()
+ Customers(int, String, String)
+ Customers(String)
+ getCustomerId() : int
+ setCustomerId(int)
+ getCusName() : String
+ setCusName(String)
+ getCusEmail() : String
+ setCusEmail(String)
+ getOrders() : LinkedList<Integer>
+ registerCustomer(): void
+ placeOrderForCustomer(LinkedList<Products>, LinkedList<Orders>)
+ showOrderHistory(): void
+ customerExists(int): boolean
+ findCustomerById(): Customers
+ addOrder(Integer): void
+ removeOrder(Integer): boolean
+ printAllCustomers(): void
+ getCustomerList() : LinkedList<Customers>
+ toString(): String
1
1
0..*
Orders
- orderId: int
- customerRef: int
- products: LinkedList<Integer>
- totalPrice: double
- orderDate: String
- status: String
- orders: static LinkedList<Orders>
- static Scanner input
+ Orders()
+ Orders(int, int, Integer[], double, String, String)
+ Orders(String)
+ getOrderId() : int
+ setOrderId(int): void
+ getCustomerRef() : int
+ setCustomerRef(int): void
+ getProducts() : LinkedList<Integer>
+ setProducts(LinkedList<Integer>): void
+ getTotalPrice() : double
+ setTotalPrice(double): void
+ getOrderDate() : String
+ setOrderDate(String): void
+ getStatus() : String
+ setStatus(String): void
+ createOrder(int, LinkedList): Orders
+ cancelOrder(int): int
+ UpdateOrder(int): boolean
+ searchById(int): Orders
+ getOrdersBetween(String, String): LinkedList
+ checkOrderID(int): boolean
+ addProduct(Integer): void
+ removeProduct(Integer): boolean
+ getAllOrders() : LinkedList<Orders>
+ toString(): String
Products
- productId: int
- name: String
- price: double
- stock: int
- reviews: LinkedList<Integer>
- static Scanner input
- products: static LinkedList<Products>
+ Products()
+ Products(int, String, double, int)
+ Products(String fileName)
+ getproductsData() : LinkedList<Products>
+ getProductId() : int
+ setProductId(int): void
+ getProductName() : String
+ setProductName(String): void
+ getProductPrice() : double
+ setProductPrice(double): void
+ getStock() : int
+ setStock(int): void
+ getReviews() : LinkedList<Integer>
+ addProduct(): void
+ addProduct(): void
+ removeProduct(): Products
+ updateProduct(): Products
+ searchProducID(): Products
+ searchProducName(): Products
+ OutStockProducts(): void
+ checkProductID(int): boolean
+ getProductData(int): Products
+ addStock(int): void
+ removeStock(int): void
+ addReview(Integer): void
+ removeReview(Integer) : boolean
1
Reviews
- reviewId: int
- productID: int
- customerID: int
- rating: int
- comment: String
- static Scanner input
- static LinkedList<Reviews> reviews
+ Reviews()
+ Reviews(int, int, int, int, String)
+ Reviews(String)
+ getReviewId() : int
+ setReviewId(int): void
+ getProduct() : int
+ setProduct(int): void
+ getCustomer() : int
+ setCustomer(int): void
+ getRating() : int
+ setRating(int): void
+ getComment() : String
+ setComment(String): void
+ AddReview(int, int): Reviews
+ updateReview(): void
+ AVG_Rating(int): float
+ checkReviewID(int): boolean
+ getReviewsByCustomer(int): LinkedList
+ getReviewData() : LinkedList<Reviews>
+ toString(): String
0..*
0..*

Performance Analysis

Method	Time Complexity	Space Complexity
AddReview(int customerID, int productID)	O(n)	O(1)
updateReview()	O(n)	O(1)
AVG_Rating(int pid)	O(n)	O(1)
addProduct()	O(n)	O(1)
removeProduct()	O(n)	O(1)
updateProduct()	O(m)	O(1)
searchProducID()	O(n)	O(1)
searchProducName()	O(n)	O(1)
OutStockProducts()	O(n)	O(1)
registerCustomer()	O(n)	O(1)
placeOrderForCustomer(LinkedList<Products> products, LinkedList<Orders> orders)	O(n)	O(n)
showOrderHistory()	O(n+ k)	O(1)
createOrder(int customerId, LinkedList<Products> productsList)	O(n × m)	O(1)
cancelOrder(int id)	O(n)	O(1)
UpdateOrder(int orderID)	O(n)	O(1)
searchById(int id)	O(n)	O(1)
extractCustomerReviews()	O(n)	O(n)
top3Products()	O(n^2)	O(n)
commonProducts(int cid1 , int cid2)	O(n* m)	O(m1 + m2)
PlaceOrder()	O(n)	O(1)
getOrdersBetween(String start, String end)	O(n)	O(n)

Frequency

- Class Reviews Operations:

AVG_Rating Method

Line	Statement	Frequency	Total
1	public static float AVG_Rating(int pid)	0	0
2	{	-	0
3	try {	0	0
4	int counter = 0;	1	1
5	float rate = 0;	1	1
6		0	0
7	reviews.findFirst();	1	1
8	while (!reviews.last())	n+1	n+1
9	{	0	0
10	if (reviews.retrieve().getProduct()==pid)	n	n
11	{	0	0
12	counter++;	n	n
13	rate+=reviews.retrieve().getRating();	n	n
14	}	-	0
15	reviews.findNext();	n	n
16	}	-	0
17	if (reviews.retrieve().getProduct()==pid)	1	1
18	{	0	0
19	counter++;	1	1
20	rate+=reviews.retrieve().getRating();	1	1
21	}	-	0
22		0	0
23	return(rate/counter);	1	1
24	} catch (Exception e) {	0	0
25	System.out.println(...);	1	1
26	return 0;	1	1
27	}	-	0
28	}	-	0

Final Total: 5n + 8

AddReview Method

Line	Statement	Frequency	Total
1	public Reviews AddReview(int customerID, int productID) {	0	0
2	{	-	0
3	try {	0	0
4	System.out.println("Please enter the review ID:");	1	1
5	reviewID = input.nextInt();	1	1
6	while(checkReviewID(reviewID))	n+1	n+1
7	{	-	0
8	System.out.println("….");	n	n
9	reviewID = input.nextInt();	n	n
10	}	-	0
11	System.out.println("….");	1	1
12	int rate = input.nextInt();	1	1
13	while(rate > 5 OR rate < 1)	n+1	n+1
14	{	-	0
15	System.out.println("….");	n	n
16	rate = input.nextInt();	n	n
17	}	-	0
18	System.out.println("….");	1	1
19	String comment = input.nextLine();	1	1
20	comment = input.nextLine();	1	1
21	Reviews review = new Reviews(reviewID, productID, customerID, rate, comment);	1	1
22	if(reviews.empty())	1	1
23	{	-	0
24	reviews.insert(review);	1	1
25	} else {	0	0
26	reviews.findFirst();	1	1
27	reviews.insert(review);	1	1
28	}	-	0
29	return review;	1	1
30	} catch(Exception e) {	0	0
31	System.out.println("….");	1	1
32	input.nextLine();	1	1
33	return null;	1	1
34	}	-	0
35	}	-	0

Final total = 6n + 13

updateReview method

Line	Statement	Frequency	Total
1	public void updateReview()	0	0
2	{	-	0
3	try {	0	0
4	if(reviews.empty())	1	1
5	System.out.println("….");	1	1
6	return;	1	1
7	}	-	0
8	System.out.println("….");	1	1
9	int reviewID = input.nextInt();	1	1
10	while(!checkReviewID(reviewID))	n+1	n+1
11	{	-	0
12	System.out.println("….");	n	n
13	reviewID = input.nextInt();	n	n
14	}	-	0
15	reviews.findFirst();	1	1
16	boolean found = false;	1	1
17	for(int i=0; i<reviews.size(); i++)	n	n
18	{	-	0
19	if(reviews.retrieve().getReviewId()==reviewID)	n	n
20	found = true;	n	n
21	break;	n	n
22	}	-	0
23	if(!reviews.last())	n	n
24	reviews.findNext();	n	n
25	}	-	0
26	if(found)	1	1
27	Reviews review = reviews.retrieve();	1	1
28	System.out.println("….");	1	1
29	System.out.println("….");	1	1
30	System.out.println("….");	1	1
31	int choice = input.nextInt();	1	1
32	switch(choice) {	1	1
33	case 1:	0	0
34	System.out.println("….");	1	1
35	int rate = input.nextInt();	1	1
36	while(rate>5 OR rate<1) {	n+1	n+1
37	System.out.println("….");	n	n
38	rate = input.nextInt();	n	n
39	}	-	0
40	review.setRating(rate);	1	1
41	break;	1	1
42	case 2:	0	0
43	System.out.println("….");	1	1
44	String comment = input.nextLine();	1	1
45	comment = input.nextLine();	1	1
46	review.setComment(comment);	1	1

47	break;	1	1
48	}	-	0
49	System.out.println("….");	1	1
50	System.out.println(reviews.retrieve());	1	1
51	}	-	0
52	} catch(Exception e) {	0	0
53	System.out.println("….");	1	1
54	input.nextLine();	1	1
55	}	-	0
56	}	-	0

Final total= 29+11n

-Class Products Operations:

addProduct Method

Line	Statement	Frequency	Total
1	public void addProduct() {	-	0
2	try {	0	0
3	System.out.println("Please enter the product ID:");	1	1
4	int productId = input.nextInt();	1	1
5	while (checkProductID(productId)) {	n+1	n+1
6	System.out.println("This ID already exists, please try a different one:");	n	n
7	productId = input.nextInt();	n	n
8	}	-	0
9	System.out.println("Enter the product name:");	1	1
10	String name = input.next();	1	1
11	System.out.println("Enter the price:");	1	1
12	double price =input.nextDouble();	1	1
13	System.out.println("stock:");	1	1
14	int stock = input.nextInt();	1	1
15	Products product = new Products(productId, name, price, stock);	1	1
16	if (products.empty()) {	1	1
17	products.insert(product);	1	1
18	} else {	0	0
19	products.findFirst();	1	1
20	products.insert(product);	1	1
21	}	-	0
22	} catch(Exception e){	0	0

23	System.out.println(" Invalid input, please try again.");	1	1
24	input.nextLine();	1	1
25	}	-	0
26	}	-	0

Final total = 3n + 15

removeProduct Method

Line	Statement	Frequency	Total
1	public Products removeProduct() {	-	0
2	try {	0	0
3	if (products.empty()) {	0	0
4	System.out.println("The product list is currently empty");	1	1
5	return null;	1	1
6	}	-	0
7	System.out.println("Enter the product ID to remove: ");	1	1
8	int productID = input.nextInt();	1	1
9	products.findFirst();	1	1
10	for (int i = 0; i < products.size(); i++) {	n+1	n+1
11	if(products.retrieve().getProductId() == productID) {	n	n
12	Products removedProduct = products.retrieve();	n	n
13	products.remove();	n	n
14	System.out.println("Product " + productID + " has been removed from the system");	n	n
15	return removedProduct;	n	n
16	}	-	0
17	if (!products.last())	n	n
18	products.findNext();	n	n
19	}	-	0
20	System.out.println("Product not found in the system");	1	1
21	return null;	1	1
22	} catch (Exception e) {	0	0
23	System.out.println(" Invalid input, please try again.");	1	1
24	input.nextLine();	1	1
25	return null;	1	1
26	}	-	0
27	}	-	0

Final total = 8n + 11

updateProduct Method

Line	Statement	Frequency	Total
1	public Products updateProduct() {	-	0
2	try{	0	0
3	if (products.empty()) {	1	1
4	System.out.println("The product list is currently empty");	1	1
5	return null;	1	1
6	}	-	0
7	System.out.println("Enter the product ID to update: ");	1	1
8	int productID = input.nextInt();	1	1
9	products.findFirst();	1	1
10	boolean found = false;	1	1
11	for (int i = 0; i < products.size(); i++) {	n+1	n+1
12	if (products.retrieve().getProductId() == productID) {	n	n
13	found = true;	n	n
14	break;	n	n
15	}	-	0
16	if (!products.last())	n	n
17	products.findNext();	n	n
18	}	-	0
19	if (found) {	1	1
20	Products p = products.retrieve();	1	1
21	System.out.println("1. Update product Name");	1	1
22	System.out.println("2. Update product price");	1	1
23	System.out.println("3. Update stock");	1	1
24	System.out.println("Enter your choice");	1	1
25	int choice = input.nextInt();	1	1
26	switch (choice) {	1	1
27	case 1:	1	1
28	System.out.println("Enter new product name:");	1	1
29	p.setProductName(input.next());	1	1
30	break;	1	1
31	case 2:	1	1
32	System.out.println("Enter new price:");	1	1
33	p.setProductPrice(input.nextDouble());	1	1
34	break;	1	1
35	case 3:	1	0
36	System.out.println("Enter new stock quantity:");	1	1

37	int stock = input.nextInt();	1	1
38	p.setStock(stock);	1	1
39	break;	1	1
40	default:	1	1
41	System.out.println("Invalid selection");	1	1
42	}	-	0
43	return p;	1	1
44	}	-	0
45	System.out.println("Product not found in the system");	1	1
46	return null;	1	1
47	} catch (Exception e) {	0	0
48	System.out.println(" Invalid input, please try again.");	1	1
49	input.nextLine();	1	1
50	retutn null;	1	1
51	}	-	0
51	}	-	0

Final total = 6n + 37

searchProducID Method

Line	Statement	Frequency	Total
1	public Products searchProducID() {	-	0
2	try {	0	0
3	if (products.empty()) {	1	1
4	System.out.println("The product list is currently empty");	1	1
5	return null;	1	1
6	}	-	0
7	System.out.println("Please enter the product ID: ");	1	1
8	int productID = input.nextInt();	1	1
9	products.findFirst();	1	1
10	for (int i = 0; i < products.size(); i++) {	n+1	n+1
11	if (products.retrieve().getProductId() == productID) {	n	n
12	return products.retrieve();	n	n
13	}	-	0
14	if (!products.last())	n	n
15	products.findNext();	n	n
16	}	n	n
17	System.out.println("Product not found in the system");	1	1
18	return null;	1	1
19	}	-	0

20	} catch(Exception e){	0	0
21	System.out.println(" Invalid input, please try again.");	1	1
22	input.nextLine();	1	1
23	return null;	1	1
24	}	-	0
25	}	-	0

Final total = 6n + 12

searchProducName Method

Line	Statement	Frequency	Total
1	public Products searchProducName() {	-	0
2	try {	0	0
3	if (products.empty()) {	1	1
4	System.out.println("The product list is currently empty");	1	1
5	return null;	1	1
6	}	-	0
7	System.out.println("Enter product Name to search: ");	1	1
8	String name = input.nextLine();	1	1
9	name = input.nextLine();	1	1
10	products.findFirst();	1	1
11	for (int i = 0; i < products.size(); i++) {	n+1	n+1
12	if (products.retrieve().getProductName().equalsIgnoreCase(name)) {	n	n
13	return products.retrieve();	n	n
14	}	-	0
15	if (!products.last())	n	n
16	products.findNext();	n	n
17	}	-	0
18	System.out.println("No product found with this name");	1	1
19	return null;	1	1
20	} catch(Exception e){	0	0
21	System.out.println(" Invalid input, please try again.");	1	1
22	input.nextLine();	1	1
23	return null;	1	1
24	}	-	0
25	}	-	0

Final total = 4n+ 13

OutStockProducts Method

Line	Statement	Frequency	Total
1	public void OutStockProducts() {	-	0
2	try {	0	0

3	if (products.empty()) {	1	1
4	System.out.println("The product list is currently empty");	1	1
5	return;	1	1
6	}	-	0
7	boolean found = false;	1	1
8	products.findFirst();	1	1
9	for (int i = 0; i < products.size(); i++) {	n+1	n+1
10	if (products.retrieve().getStock() == 0) {	n	n
11	System.out.println(products.retrieve());	n	n
12	found = true;	n	n
13	}	-	0
14	if (!products.last())	n	n
15	products.findNext();	n	n
16	}	-	0
17	if (!found) {	1	1
18	System.out.println("There is NO product out of stock");	1	1
19	}	-	0
20	} catch (Exception e) {	0	0
21	System.out.println(" Error occurred while checking stock, please try again.");	1	1
22	}	-	0
23	}	-	0

Final total = 6n+ 9

-Class Customers Operations:

registerCustomer() Method

Line	Statement	Frequency	Total
1	public void registerCustomer() {	-	0
2	try {	0	0
3	System.out.print("Enter Customer ID: ");	1	1
4	int id = input.nextInt();	1	1
5	input.nextLine();	1	1
6	while (customerExists(id)) {	n+1	n+1
7	System.out.println("This ID already exists. Try again.");	n	n
8	id = input.nextInt();	n	n
9	input.nextLine();	n	n
10	}	-	0
11	System.out.print("Enter Customer Name: ");	1	1
12	String name = input.nextLine();	1	1
13	System.out.print("Enter Customer Email: ");	1	1
14	String email = input.nextLine();	1	1
15	Customers c = new Customers(id, name, email);	1	1
16	if (customers.empty()) {	1	1
17	customers.insert(c);	1	1
18	} else {	0	0

19	customers.findFirst();	1	1
20	customers.insert(c);	1	1
21	}	-	0
22	System.out.println("Customer added successfully.");	1	1
23	} catch(Exception e){	0	0
24	System.out.println(" Invalid input, please try again.");	1	1
25	input.nextLine();	1	1
26	}	-	0
27	}	-	0

Final total = 4n+ 16

placeOrderForCustomer Method

Line	Statement	Frequency	Total
1	public void placeOrderForCustomer(LinkedList<Products> products, LinkedList<Orders> orders) {	-	0
2	try {	0	0
3	Customers customer = findCustomerById();	1	1
4	if (customer == null) {	1	1
5	System.out.println("Customer not found. Cannot place order.");	1	1
6	return;	1	1
7	}	-	0
8	Orders orderHandler = new Orders();	1	1
9	Orders newOrder = orderHandler.createOrder(customer.getCustomerId(), products);	1	1
10	if (newOrder != null) {	1	1
11	customer.addOrder(newOrder.getOrderId());	1	1
12	System.out.println("Order successfully placed for customer " + customer.getCustomerId());	1	1
13	System.out.println(newOrder);	1	1
14	}	-	0
15	} catch (Exception e) {	0	0
16	System.out.println("An unexpected error occurred while placing the order. Please try again.");	1	1
17	}	-	0
18	}	-	0

Final total = 11

showOrderHistory Method

Line	Statement	Frequency	Total
1	public void showOrderHistory() {	-	0
2	try {	0	0
3	if (customers.empty()) {	1	1
4	System.out.println("No customers in the system.");	1	1
5	return;	1	1

6	}	-	0
7	Customers target = findCustomerById();	1	1
8	if (target == null) {	1	1
9	System.out.println("Customer not found.");	1	1
10	return;	1	1
11	}	-	0
12	LinkedList<Integer> orderList = target.getOrders();	1	1
13	if (orderList.empty()) {	1	1
14	System.out.println("No orders found for customer.");	1	1
15	return;	1	1
16	}	-	0
17	System.out.println("Order History for Customer " + target.getCustomerId() + ":");	1	1
18	orderList.findFirst();	1	1
19	while (!orderList.last()) {	n+1	n+1
20	System.out.println("Order ID: " + orderList.retrieve());	n	n
21	orderList.findNext();	n	n
22	}	-	0
23	System.out.println("Order ID: " + orderList.retrieve());	1	1
24	} catch (Exception e) {	0	0
25	System.out.println(" Error occurred while Showing Order history, please try again.");	1	1
26	}	-	0
27	}	-	0

Final total = 3n+ 16

-Class Orders Operations:

CreateOrder Method

Line	Statment	Frequency	Total
1	public Orders createOrder(int customerId, LinkedList<Products> productsList) {	0	0
2	try {	0	0
1	Orders newOrder = new Orders();	1	1
2	int totalPrice = 0;	1	1
3	System.out.prtinln("….");	1	1
4	int oid = input.nextInt();	1	1
5	while (checkOrderID(oid)) {	k	k
6	System.out.prtinln("….");	k-1	k-1
7	oid = input.nextInt(); }	k-1	k-1
8	newOrder.setOrderId(oid);	1	1
9	newOrder.setCustomerRef(customerId);	1	1
10	char answer='y';	1	1
11	while (answer == 'y' || answer == 'Y') {	n+1	n+1
12	System.out.prtinln("….");	n	n
13	int pid = input.nextInt();	n	n

14	while (!checkProductExists(pid, productsList)) {	np	np
15	System.out.prtinln("….");	n(p-1)	n(p-1)
16	pid=input.nextInt(); }	n(p-1)	n(p-1)
17	productsList.findFirst();	n	n
18	for (int i = 0; i < productsList.size(); i++) {	nm	nm
19	if (productsList.retrieve().getProductId() == pid) {	nm	nm
20	if (productsList.retrieve().getStock() == 0) {	n	n
21	System.out.prtinln("….");	s	s
22	} else {	0	0
23	productsList.retrieve().setStock(productsList.retrieve().getStock() - 1);	n-s	n-s
24	System.out.prtinln("….");	n-s	n-s
25	newOrder.addProduct(productsList.retrieve().getProductId());	n-s	n-s
26	totalPrice += productsList.retrieve().getProductPrice(); }	n-s	n-s
27	break; }	n	n
28	if (!productsList.last())	n(m-1)	n(m-1)
29	productsList.findNext();	n(m-1)	n(m-1)
30	System.out.prtinln("….");	n	n
31	answer = input.next().charAt(0);	n	n
32	newOrder.setTotalPrice(totalPrice);	1	1
33	boolean validDate = false;	1	1
34	String formattedDate = "";	1	1
35	while (!validDate) {	d+1	s+1
36	try {	0	d
37	System.out.prtinln("….");	d	d
38	String orderDateInput = input.next();	d	d
39	DateTimeFormatter inputFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");	d	d
40	DateTimeFormatter outputFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");	d	d
41	LocalDate date = LocalDate.parse(orderDateInput, inputFormatter);	d	d
42	formattedDate = date.format(outputFormatter);	d	d
43	validDate=true;	1	1
44	System.out.prtinln("….");	1	1
45	} catch (Exception e) {	d-1	d-1
46	System.out.prtinln("…."); } }	d-1	d-1
47	newOrder.setOrderDate(formattedDate);	1	1
48	String status;	1	1
49	boolean validStatus = false;	1	1
50	do {	t+1	t+1
51	System.out.prtinln("….");	t	t
52	status = input.next().trim().toLowerCase();	t	t
53	if (status.equals("pending") || status.equals("shipped") || status.equals("delivered") || status.equals("cancelled")) {	t	t
54	validStatus = true;	t	t
55	} else {	0	0
56	System.out.prtinln("…."); }	1	1
57	} while (!validStatus);	S	1

58	newOrder.setStatus(status);	1	1
59	orders.insert(newOrder);	1	1
60	System.out.prtinln("….");	1	1
61	return newOrder;	1	1
62	} catch (Exception e) {	0	0
63	System.out.prtinln("….");	1	1
64	return null; } }	1	1

Fianl total = 15 + 3k + (n+1) + 4n + 3n×p + n×m + 3n×m + 7(n-s) + s + 8d + 5t

cancelOrder(int id) Method

Line	Statement	Frequency	Total
1	public int cancelOrder(int id){	0	0
2	try {	0	0
3	if (orders.empty()){	1	1
4	System.out.println("No orders available");	1	1
5	return 0;	1	1
6	}	-	0
7	orders.findFirst();	1	1
8	for (i = 0; i < orders.size(); i++)	n+1	n
9	if (retrieve().getOrderId() == id)	n	n
10	if (status == "cancelled")	n	n
11	System.out.println("….");	n	n
12	return 2;	n	n
13	}	-	0
14	orders.retrieve().setStatus("cancelled");	n	n
15	System.out.println("….")	n	n
16	return 1;	n	n
17	}	-	0
19	if (!orders.last())	n	n
20	orders.findNext();	n	n
21	}	-	0
22	System.out.println("Order " + id + " not found.");	1	1
23	return 0;	1	1
24	} catch (Exception e) {		
25	System.out.println("….");	1	1
26	return 0;	1	1
27	}	-	0
28	}	-	0

Final total = 10n+9

UpdateOrder(int orderID) Method

Line	Statement	Frequency	Total
1	public boolean UpdateOrder(int orderID)	-	0
2	try {	0	0
3	if (orders.empty())	1	1
4	System.out.println("….");	1	1
5	return false;	1	1
6	}	-	0
7	orders.findFirst();	1	1
8	boolean found = false;	1	1
10	for (i = 0; i < orders.size(); i++)	m+1	m+1
11	if (retrieve().getOrderId() == orderID)	m	m
12	found = true;	1	1
13	break;	1	1
14	}	-	0
15	if (!orders.last())	m-1	m-1
16	orders.findNext();	m-1	m-1
17	}	-	0
18	if (found) {	1	1
19	Orders current = orders.retrieve();	1	1
20	if (current.getStatus().equalsIgnoreCase("cancelled")) {	1	1
21	System.out.println("….");	1	1
22	return false;	1	1
23	}	-	0
24	System.out.println("….");	1	1
25	String[] validStatuses = {"pending", "shipped", "delivered", "cancelled"};	1	1
26	String newStatus;	1	1
27	while (true) {	s+1	s+1
28	System.out.println("….");	s	s
29	newStatus = input.next().toLowerCase();	s	s
30	}	-	0
31	System.out.println("….");	1	1
32	boolean isValid = false;	s	s
33	for (String status : validStatuses) {	s×4	s×4
34	if (newStatus.equalsIgnoreCase(status)) {	s	s
35	isValid = true;	s	s
36	break; } }	s	s
37	if (isValid) {	-	0
38	break;	-	0
39	} else {	s-1	s-1
40	System.out.println("…."); } }	1	1
41	current.setStatus(newStatus);	1	1
42	System.out.println("….");	1	1
43	return true; }	1	1
44	System.out.println("….");	1	1
45	return false;	1	1
46	} catch (Exception e) {	0	0
47	System.out.println("….");	1	1

48	input.nextLine();	1	1
49	return false; } }	1	1

Final total =4m+12s+24

searchById(int id) Method

Line	Statement	Frequency	Total
1	public static Orders searchById(int id)	0	0
2	try {	0	0
3	if (orders.empty())	1	1
4	return null;	1	1
6	orders.findFirst();	1	1
7	for (i=0; i<orders.size(); i++) {	n+1	n+1
8	if (orders.retrieve().getOrderId()==id) {	n	n
9	return orders.retrieve();	n	n
10	}	-	0
11	if (!orders.last())	n	n
12	orders.findNext();	n	n
13	}	-	0
14	System.out.println("….");	1	1
15	return null;	1	1
16	} catch (Exception e) {	0	0
17	System.out.println("….");	1	1
18	return null;	1	1
19	}	-	0
20	}	-	0

Final total = 5n+8

- User-level system functionalities:

extractCustomerReviews() Method

Line	Statment	Frequency	Total
1	public static void extractCustomerReviews() {	0	0
2	try {	0	0
3	System.out.println("….");	1	1
4	int customerId = input.nextInt();	1	1
5	while (!cdata.customerExists(customerId)) {	n+1	n+1
6	System.out.println("….");	n	n
7	customerId = input.nextInt();	n	n
	}	-	0
8	LinkedList<Reviews> customerReviews = rdata.getReviewsByCustomer(customerId);	1	1
9	if (customerReviews.empty()) {	1	1
10	System.out.println("….");	1	1
11	} else {	0	0

12	System.out.println("….");	1	1
13	System.out.println("….");	1	1
14	System.out.println("….");	1	1
15	customerReviews.findFirst();	1	1
16	for (int i = 0; i < customerReviews.size(); i++) {	n+1	n+1
17	Reviews r = customerReviews.retrieve();	n	n
18	System.out.println("….");	n	n
19	System.out.println("….");	n	n
20	System.out.println("….");	n	n
21	System.out.println("….");	n	n
22	System.out.println("….");	n	n
23	if (!customerReviews.last())	n	n
24	customerReviews.findNext();	n	n
	}	-	0
25	System.out.println("….");	1	1
26	} catch (Exception e) {	0	0
27	System.out.println("….");	1	1
28	}	-	0

Final total = 12n+13

commonProducts Method

Line	Statement	Frequency	Total
1	public static void commonProducts(int cid1 , int cid2)	-	0
2	{	-	0
3	try{	1	1
4	LinkedList<Integer> pcustomer1 = new LinkedList<Integer> ();	1	1
6	LinkedList <Integer> pcustomer2 = new LinkedList <Integer> ();	1	1
7	reviews = rdata.getReviewData();	1	1
8	if (! reviews.empty())	1	1
9	{	-	0
10	reviews.findFirst();	1	1
11	for (int i =1 ;i <= reviews.size() ; i++)	n+1	n+1
12	{	-	0
13	if (reviews.retrieve().getCustomer() == cid1)	n	n
14	{	-	0
15	pcustomer1.findFirst();	n	n
16	boolean found1 = false;	n	n
17	for (int x = 1; x <= pcustomer1.size() ; x++)	n*m+1	nm+n
18	{	-	0
19	if (pcustomer1.retrieve() == reviews.retrieve().getProduct())	n*m	nm
20	{	-	0
21	found1 = true;	n*m	nm
22	break;	n*m	nm

23	}	-	0
24	pcustomer1.findNext();	n*m	nm
25	}	-	0
26	pcustomer1.findFirst();	n	n
27	if (! found1)	n	n
28	pcustomer1.insert(reviews.retrieve().getProduct());	n	n
29	}	-	0
30	if (reviews.retrieve().getCustomer() == cid2)	n	n
31	{	-	0
32	pcustomer2.findFirst();	n	n
33	boolean found2 = false;	n	n
34	for (int x = 1; x <= pcustomer2.size() ; x++)	n*k+1	nk+n
35	{	-	0
36	if (pcustomer2.retrieve() == reviews.retrieve().getProduct())	n*k	nk
37	{	-	0
38	found2 = true;	n*k	nk
39	break;	n*k	nk
40	}	-	0
41	pcustomer2.findNext();	n*k	nk
42	}	-	0
43	pcustomer2.findFirst();	n	n
44	if (! found2)	n	n
45	pcustomer2.insert(reviews.retrieve().getProduct());	n	n
46	}	-	0
47	reviews.findNext();	n	n
48	}	-	0
49	pcustomer1.print();	1	1
50	pcustomer2.print();	1	1
51	PQ_LinkedList<Products> AVGrate45 = new PQ_LinkedList<Products> ();	1	1
52	if (! pcustomer1.empty() && ! pcustomer2.empty())	1	1
53	{	-	0
54	pcustomer1.findFirst();	1	1
55	for (int m =1; m <= pcustomer1.size() ; m++)	m+1	m+1
56	{	-	0
57	int pID = pcustomer1.retrieve();	m	m
58	pcustomer2.findFirst();	m	m
59	for (int n = 1 ; n <= pcustomer2.size() ; n++)	m*n+1	mn+m
60	{	-	0
61	if (pID == pcustomer2.retrieve())	m*n	mn
62	{	-	0
63	float AVGrating = Reviews.AVG_Rating (pID);	m*n	mn
64	if (AVGrating > 4)	m*n	mn
65	{	-	0
66	Products p = pdata.getProductData(pID);	m*n	mn
67	AVGrate45.enqueue(p, AVGrating);	m*n	mn
68	}	-	0
69	}	-	0

70	pcustomer2.findNext();	m*n	mn
71	}	-	0
72	pcustomer1.findNext();	m	m
73	}	-	0
74	System.out.println("Common Products with rate above 4 are ");	1	1
75	while (AVGrate45.length() > 0)	n+1	n+1
76	{	-	0
77	PQ_Element<Products> product_rate = AVGrate45.serve();	n	n
78	System.out.println(" Product (" + product_rate.data.getProductId() + ") " + product_rate.data.getProductName() + " with rate " + product_rate.priority);	n	n
79	System.out.println(product_rate.data);	n	n
80	System.out.println("\n");	n	n
81	}	-	0
82	}	-	0
83	else	0	0
84	System.out.println("NO COMMON products between the two customers ");	1	1
85	}	-	0
86	else	0	0
87	System.out.println("Reviews not available for all products");	1	1
88	} catch (Exception e) {	0	0
89	System.out.println("An error occurred while finding common products. Please try again.");	1	1
90	}	-	0
91	}	-	0

Final total = 11mn + 2m + 5nk+ 20n + 18

Top3Products() Method

Line	Statment	Frequency	Total
1	public static void top3Products() {	-	0
2	try {	0	0
3	PQ_LinkedList<Products> top3 = new PQ_LinkedList<Products> ();	1	1
4	if (!products.empty()) {	1	1
5	products.findFirst()	1	1
6	for (int i = 0 ; i < products.size() ; i++) {	n+1	n+1
7	Products p = products.retrieve();	n	n
8	float AVGrating = Reviews.AVG_Rating (p.getProductId());	n*n	n^2
9	top3.enqueue(p, AVGrating);	n	n

10	if (!products.last())	n	n
11	products.findNext(); } }	n	n
12	System.out.println("….")	1	1
13	if (top3.length() == 0) {	1	1
14	System.out.println("…..");	1	1
15	return;	1	1
16	}	-	0
17	for (int j = 1 ; j <= 3 && top3.length() > 0 ; j++) {	4	4
18	PQ_Element<Products> top = top3.serve();	3	3
19	System.out.println("….") }	3	3
20	} catch (Exception e) {	0	0
21	System.out.println("….")	1	1
22	e.printStackTrace();	1	1
23	}	-	0
24	}	-	0

Final total = n^2 + 5n + 20

PlaceOrder() Method

Line	Statment	Frequency	Total
	public static void PlaceOrder() {	0	0
	try {	0	0
1	System.out.println("….");	1	1
2	int cid = input.nextInt()	1	1
3	while (!cdata.customerExists(cid)) {	n+1	n+1
4	System.out.println("….");	n	n
5	cid = input.nextInt(); }	n	n
6	Orders newOrder = odata.createOrder(cid, products);	1	1
7	if (newOrder != null) {	1	1
8	customers.findFirst();	1	1
9	for (int i = 0; i < customers.size(); i++) {	n + 1	n + 1
10	if (customers.retrieve().getCustomerId() == newOrder.getCustomerRef()) {	n	n
11	Customers cust = customers.retrieve();	n	n
13	customers.remove();	n	n
14	cust.addOrder(newOrder.getOrderId());	n	n
15	customers.insert(cust);	n	n
16	break; }	1	1
17	if (!customers.last())	n	n
18	customers.findNext(); } }	n	n
19	} catch (Exception e) {	0	0
	System.out.println("….");	1	1
	} }	-	0

Final total = 11n +9

getOrdersBetween Method

Line	Statment	Frequency	Total
1	public LinkedList<Orders> getOrdersBetween(String start, String end) {	-	0
2	try {	0	0
3	LinkedList<Orders> result = new LinkedList<>();	1	1
4	if (orders.empty()) {	1	1
5	System.out.println("No orders in system.");	1	1
6	return result;	1	1
7	}	-	0
	System.out.println("\n Searching for orders between: " + start + " and " + end);	1	1
8	int foundCount = 0;	1	1
9	orders.findFirst();	1	1
10	for (int i = 0; i < orders.size(); i++) {	n+1	n+1
11	Orders current = orders.retrieve();	n	n
12	if (current.getOrderDate().compareTo(start) >= 0 && current.getOrderDate().compareTo(end) <= 0) {	n	n
13	if (result.empty()) {	n	n
14	result.insert(current);	n	n
15	}else {	0	0
16	result.findFirst();	n	n
17	result.insert(current);	n	n
18	}	-	0
19	foundCount++;	n	n
20	}	-	0
21	if (!orders.last())	n	n
22	orders.findNext();	n	n
23	}	-	0
24	System.out.println("\n Total orders found: " + foundCount);	n	n
	return result;	-	0
25	} catch (Exception e) {	0	0
26	System.out.println("Error occurred while retrieving orders between dates. Please try again.");	0	0
27	return new LinkedList<Orders>();	1	1
28	}	-	0
29	}	-	0

Final total = 11n +9

Conclusion

This project successfully implemented an E-Commerce Inventory & Order Management System, demonstrating the effective application of data structures to solve business challenges.

The system utilizes the **LinkedList** structure to efficiently manage core data entities: **Products, Customers, Orders, and Reviews**. It strategically employs the LinkedList both for entity storage and for modeling complex relationships, such as linking orders to customers and reviews to products.

A key analytical feature is the use of the **PQ_LinkedList (Priority Queue)** to efficiently extract and rank the **Top 3 Products** based on their calculated average customer rating. This provides essential, prioritized insight for business decision-making.
In essence, the project delivers a robust platform that combines flexible data management (via LinkedList) with optimized analytical ranking (via PQ_LinkedList), forming a solid foundation for managing e-commerce operations.