Gebze Technical University

Computer Engineering Department

2022 Spring CSE222

Group 6

FINAL REPORT

Github Link :

https://github.com/ogzmtl/CSE222-e-Shopping-System/tree/Multifile-Edition

1. Group Members

- OĞUZ MUTLU	1801042624
- MUHAMMED İKBAL YAZICI	1801042607
- UMUT CAN ÖZAY	1901042641
- MERT GÜRŞİMŞİR	1901042646
- ENES ABDÜLHALİK	1901042803
- SALİH TANGEL	171044071

2. Problem Definition

Today, online shopping has become one of the most important parts of our lives and has radically changed our trading habits. We see many beneficial results of this changing shopping habit, such as enabling the 24/7 shopping process to be active, enabling us to access and review many more types and quantities of products in a shorter time, and using our time and energy more efficiently. Especially since 2020, with the pandemic conditions, which is the biggest reality that has changed our lives, classical shopping has almost completely left its place to online shopping. If we talk in numbers, 1 out of every 4 people living in the world today shop online.

When we look at Turkey, we see that it has the potential to reach much higher volumes in the future with its 38% growth rate. The biggest requirement for the utilization of this potential is to provide the end user with the most comfortable shopping experience possible by establishing an e-commerce system that can be easily understood and used by everyone. The biggest obstacle in front of this is that the confusion in the customer-product-store triangle makes it difficult for the customer to make a decision and continues to use the classical shopping methods she/he is used to, feeling that she/he cannot find what she/he is looking for.

As a solution to this problem, we are introducing a product-based complexity removal system rather than a store-based complexity-enhancing system. In this project, instead of a system in which the stores expand the product pool by revealing their products, the stores choose the products they sell from the existing product pool, so that the customers can clearly identify the product they want and choose the store that sells that product instead of being confused between similar products in different stores. We are creating a system where people can shop without confusion.

In this system, users will be able to list products according to their store, price, and availability while shopping through the accounts they will create. By uploading money to the system with the wallet feature, they will be able to make payments and receive refunds without the need to enter payment instrument information again until their balance reaches zero. They will be able to add the product they have chosen to their baskets and keep them waiting for purchase whenever they want, and they will be able to follow the status of their orders. Stores, on the other hand, will be able to update their product lists by choosing from the general product pool, track the orders received and the products in their stock, and view statistics such as the monthly and annual sales amount of their own stores. Thus, the project will provide the customer with a comfortable product-oriented shopping opportunity, while helping the stores to provide better service to the customers.

3. Users of the System

3.1 Customers

- People who use the app to shop and give orders by using a shopping basket and their wallet (forgotten things in the proposal are added).

3.2 Sellers (Stores)

- Firms that want to market their products on an e-shopping platform. These firms can also see statistics of their sales, send product requests to administrators, and confirm customers' orders (forgotten things in the proposal are added).

3.3 Administrators

- The persons who manage the system and makes sure everything is working fine.

4. Requirements in Details

4.1 Non-functional Requirements

- Java (jdk 17) as a programming language.
- The system runs offline.
- Bug/Log report system to keep track of what types of errors occurred.
- The system will be scalable and maintainable.
- Linux is used as Operating System
  (Forgotten things are added)

4.2 Functional Requirements

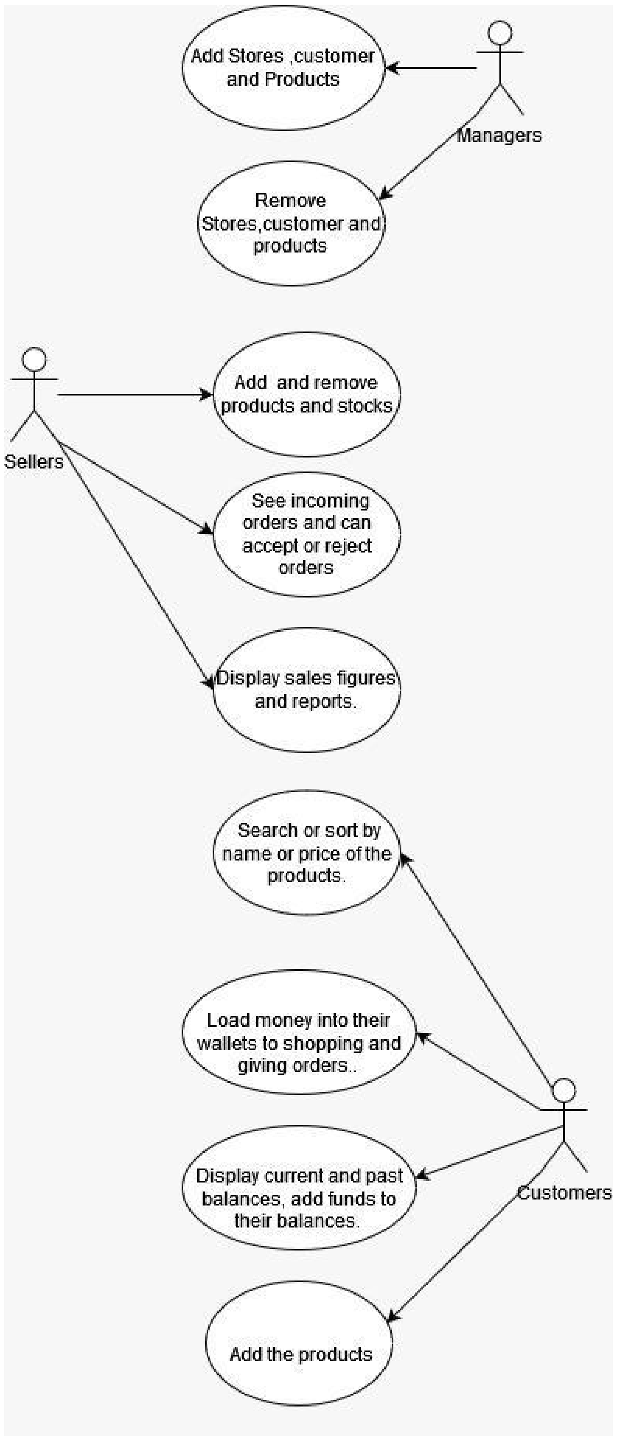
- All users can access the system using a username and a password.
- All users should be able to register by giving their information to the system.
- Managers should be able to register new stores to the system or remove stores from the system.
- Administrators will be assigned by the system.
- The system must provide the products to all sellers that use the system.
- The system has a mechanism to help recover a user's password.
- All products have a digital ID which will be generated by the system.

4.3 User Requirements

- Sellers can add new products or stocks to their stores or can remove them easily in the system.
- The seller can see the incoming orders and can accept or reject the orders.
- Sellers can display sales figures and reports.
- Customers can search or sort by name or price of the products. And they can also search and inquire about the stores.
- Users can display their previous activities.

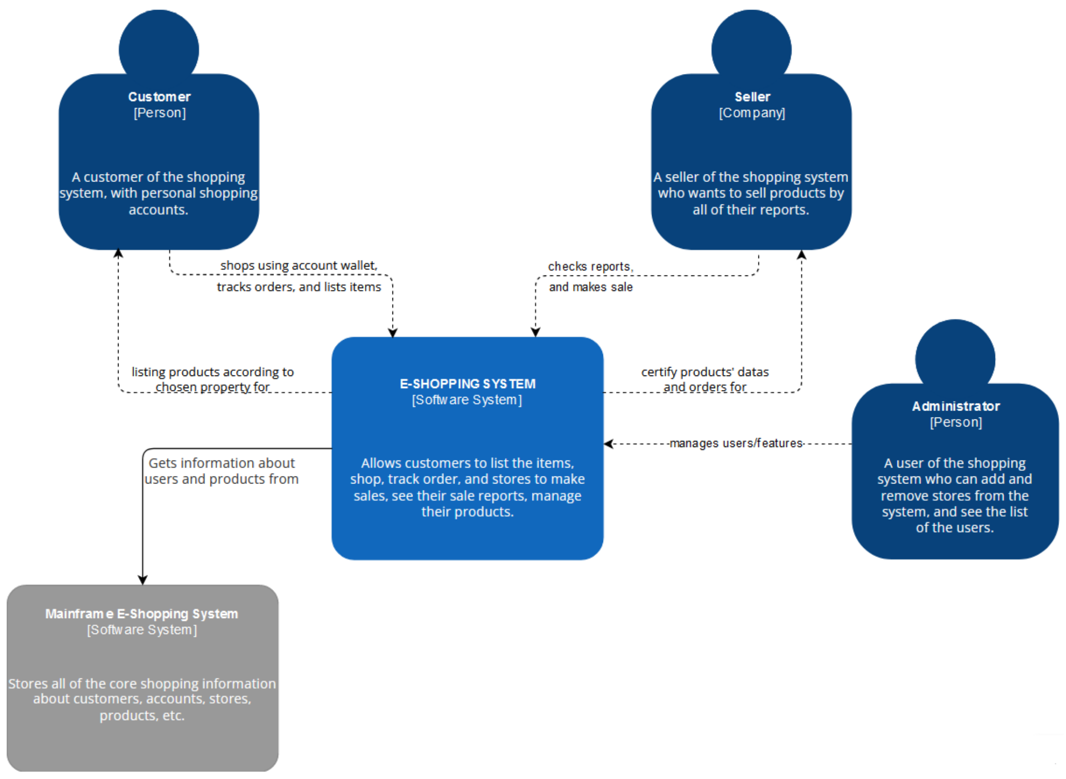
- Customers will be able to load money into their wallets to shop and give orders.
- Customers can display their current and past balances, add funds to their balances.
- Customers will be able to add the products they will buy to the cart.
- Customers can give an order by using the wallet system in the application.
- Administrators can manage the system by getting involved in the system's activities if they have to.
- Administrators can add, remove stores, customers and products.

5. Use Case Diagrams

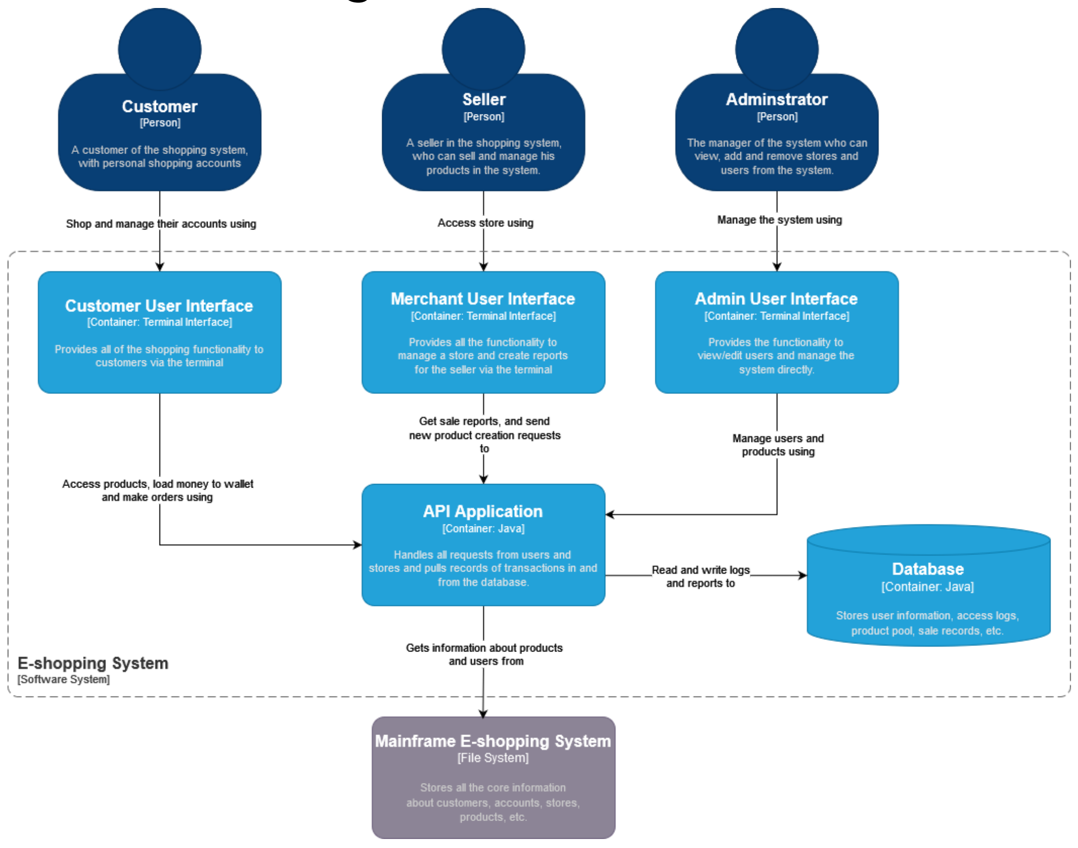

Add Stores ,customer and Products
Managers
Remove Stores,customer and products
Sellers
Add and remove products and stocks
See incoming orders and can accept or reject orders
Display sales figures and reports.
Search or sort by name or price of the products.
Load money into their wallets to shopping and giving orders..
Display current and past balances, add funds to their balances.
Customers
Add the products

6. C4 Model of The System

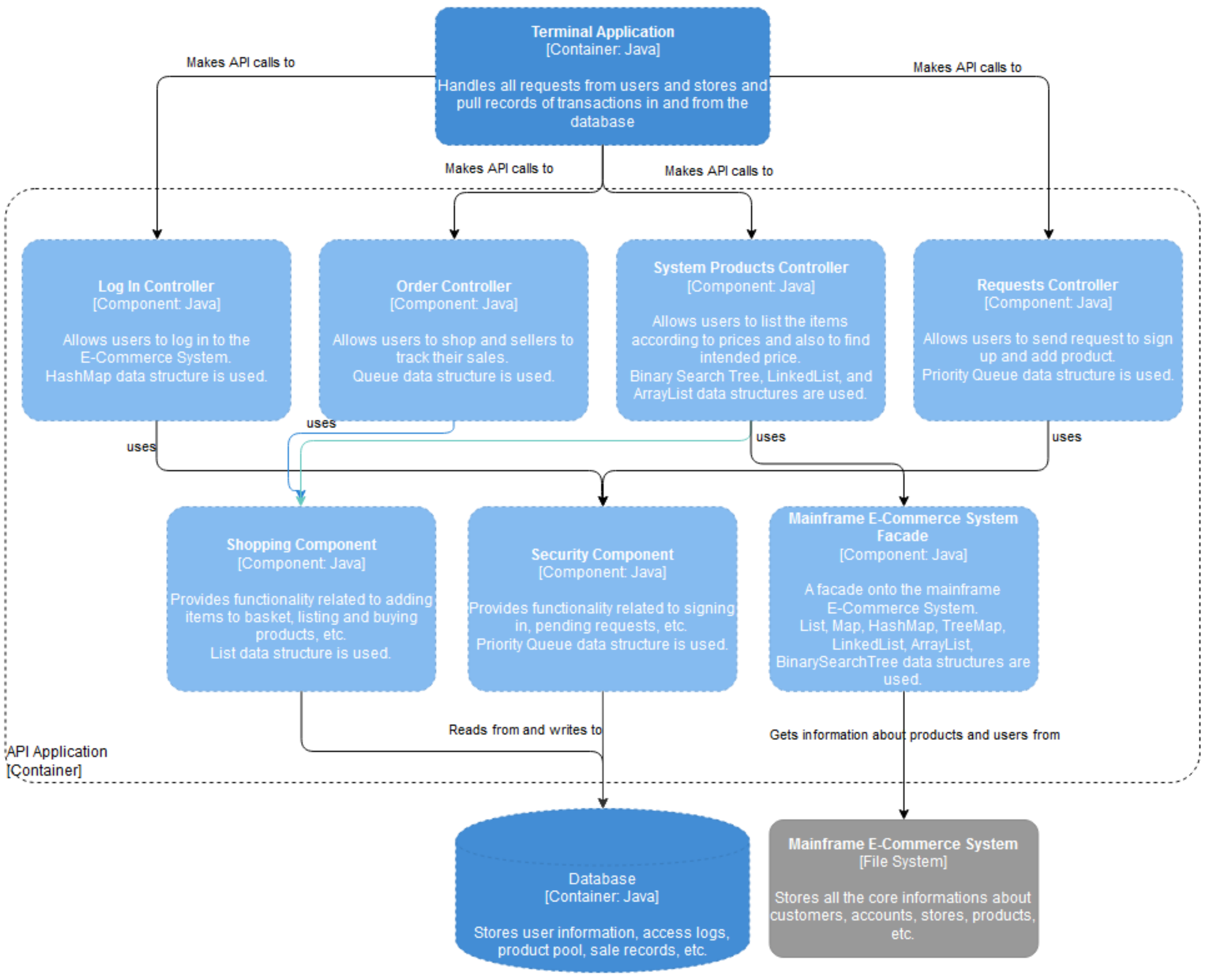
6.1 System Context Diagram


Customer
[Person]
A customer of the shopping system, with personal shopping accounts.
Seller
[Company]
A seller of the shopping system who wants to sell products by all of their reports.
shops using account wallet, tracks orders, and lists items
checks reports, and makes sale
listing products according to chosen property for
E-SHOPPING SYSTEM
[Software System]
Allows customers to list the items, shop, track order, and stores to make sales, see their sale reports, manage their products.
certify products' datas and orders for
Administrator
[Person]
manages users/features
A user of the shopping system who can add and remove stores from the system, and see the list of the users.
Gets information about users and products from
Mainframe E-Shopping System
[Software System]
Stores all of the core shopping information about customers, accounts, stores, products, etc.

6.2 Container Diagram


Customer
[Person]
A customer of the shopping system, with personal shopping accounts
Seller
[Person]
A seller in the shopping system, who can sell and manage his products in the system.
Adminstrator
[Person]
The manager of the system who can view, add and remove stores and users from the system.
Shop and manage their accounts using
Access store using
Manage the system using
Customer User Interface
[Container: Terminal Interface]
Provides all of the shopping functionality to customers via the terminal
Merchant User Interface
[Container: Terminal Interface]
Provides all the functionality to manage a store and create reports for the seller via the terminal
Admin User Interface
[Container: Terminal Interface]
Provides the functionality to view/edit users and manage the system directly.
Get sale reports, and send new product creation requests to
Manage users and products using
Access products, load money to wallet and make orders using
API Application
[Container: Java]
Handles all requests from users and stores and pulls records of transactions in and from the database.
Read and write logs and reports to
Database
[Container: Java]
Stores user information, access logs, product pool, sale records, etc.
Gets information about products and users from
E-shopping System
[Software System]
Mainframe E-shopping System
[File System]
Stores all the core information about customers, accounts, stores, products, etc.

6.3 Component Diagram


Terminal Application
[Container: Java]
Handles all requests from users and stores and pull records of transactions in and from the database
Makes API calls to
Makes API calls to
Makes API calls to
Makes API calls to
Log In Controller
[Component: Java]
Allows users to log in to the E-Commerce System.
HashMap data structure is used.
Order Controller
[Component: Java]
Allows users to shop and sellers to track their sales.
Queue data structure is used.
System Products Controller
[Component: Java]
Allows users to list the items according to prices and also to find intended price.
Binary Search Tree, LinkedList, and ArrayList data structures are used.
Requests Controller
[Component: Java]
Allows users to send request to sign up and add product.
Priority Queue data structure is used.
uses
uses
uses
uses
Shopping Component
[Component: Java]
Provides functionality related to adding items to basket, listing and buying products, etc.
List data structure is used.
Security Component
[Component: Java]
Provides functionality related to signing in, pending requests, etc.
Priority Queue data structure is used.
Mainframe E-Commerce System Facade
[Component: Java]
A facade onto the mainframe E-Commerce System.
List, Map, HashMap, TreeMap, LinkedList, ArrayList, BinarySearchTree data structures are used.
Reads from and writes to
Gets information about products and users from
API Application
[Container]
Database
[Container: Java]
Stores user information, access logs, product pool, sale records, etc.
Mainframe E-Commerce System
[File System]
Stores all the core informations about customers, accounts, stores, products, etc.

7. How Data Structures Are Used?

7.1 HashMap :

This data structure is used to log in to the system. It keeps username as key and password as the related value.

7.2 Map - LinkedList :

This map is used to keep products by product name as key and LinkedList which consists of Product objects as value. Thanks to LinkedList, products can be kept by all their information such as seller, price, etc. and also products can be removed from head or tail easily.

7.3 BinarySearchTree :

This data structure is used to keep the products. Tree is created according to the prices of a product to easily find the intended price.

7.4 ArrayList :

This data structure is used to keep the BinarySearchTrees which consists of products.

7.5 Queue :

This data structure is used to keep the orders for the seller. The seller can easily poll the last order thanks to the FIFO property of the queue.

7.6 PriorityQueue :

This data structure is used to keep requests. Requests can be sign up requests coming from the sellers or product requests to add to the general product pool. Sign up requests have higher priority than the product requests. So, the administrator of the system can easily see the top request and do whatever he/she wants.

7.7 Graph :

This data structure is used to keep products as vertices by their ID values. For this implementation we have used matrix graph because we need to access edges faster. Two products have edge between them if and only if order for that product is not canceled. So edges exist between 2 vertices if the order of the products in both vertices are either in progress or finished.

7.8 Balanced Binary Search Tree :

This data structure is used to keep products. Since we need to search a product faster, a balanced binary search tree is useful to use.

7.9 Set - Map (Navigable) :

We have used TreeMap for this structure. This data structure is used to keep the name of the products together with LinkedList which keeps sellers of each product. We have used this to access products and sellers faster and also traverse through the elements in a way we want.

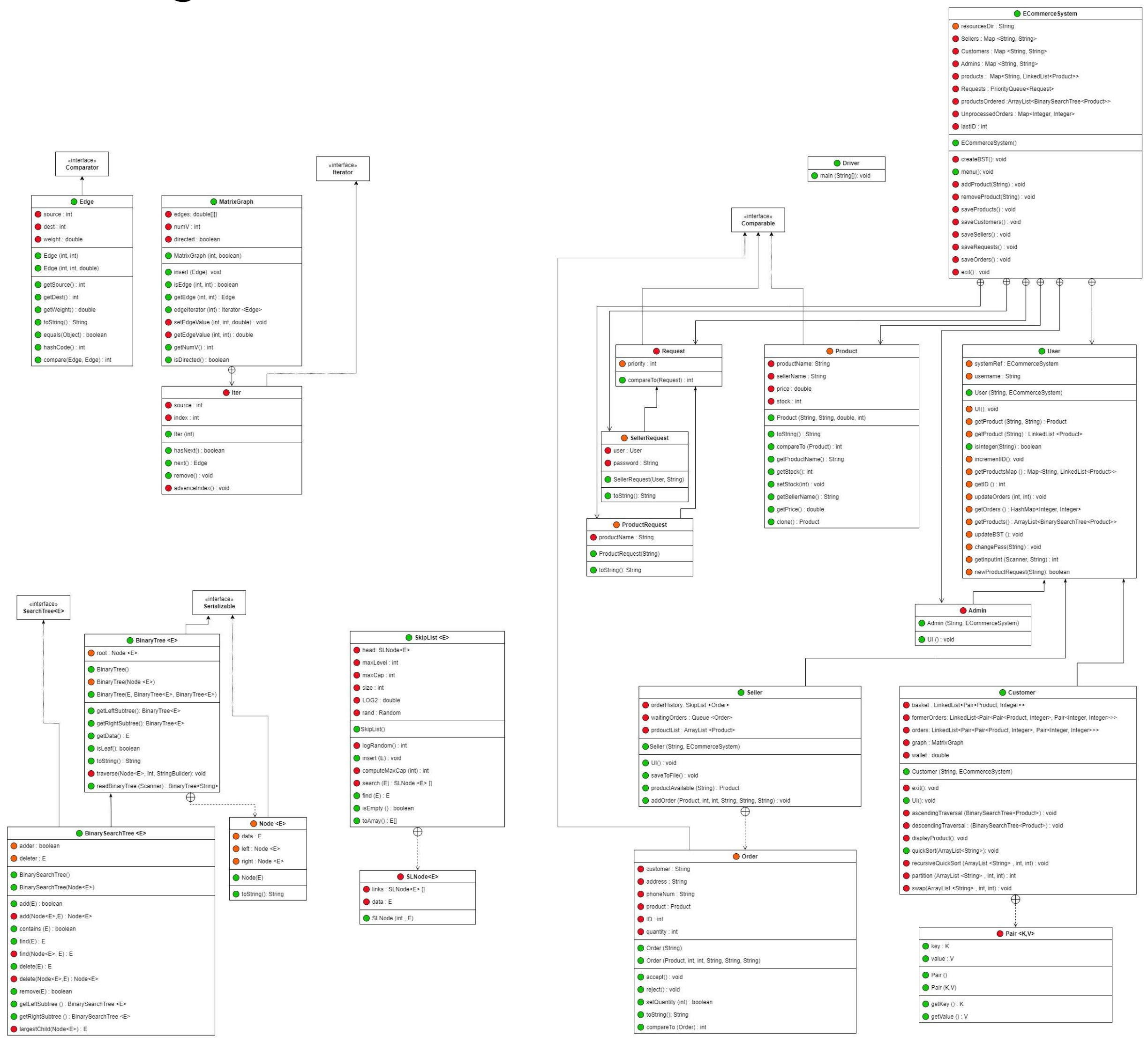
7.10 Skip List :

Being easily traversable for displaying its contents and has fast search, insertion and deletion operations, a skiplist was used to store the history of orders that were given to a seller in the seller class.

7.11 Sorting :

We have implemented and used Quick Sort as a sorting algorithm to sort products by their prices. The reason is there is a low probability to have the worst case (all the products have to have the same price and also all the sellers have to give the same price for the worst case). Since we can have too many products, quick sort is the best choice for us.

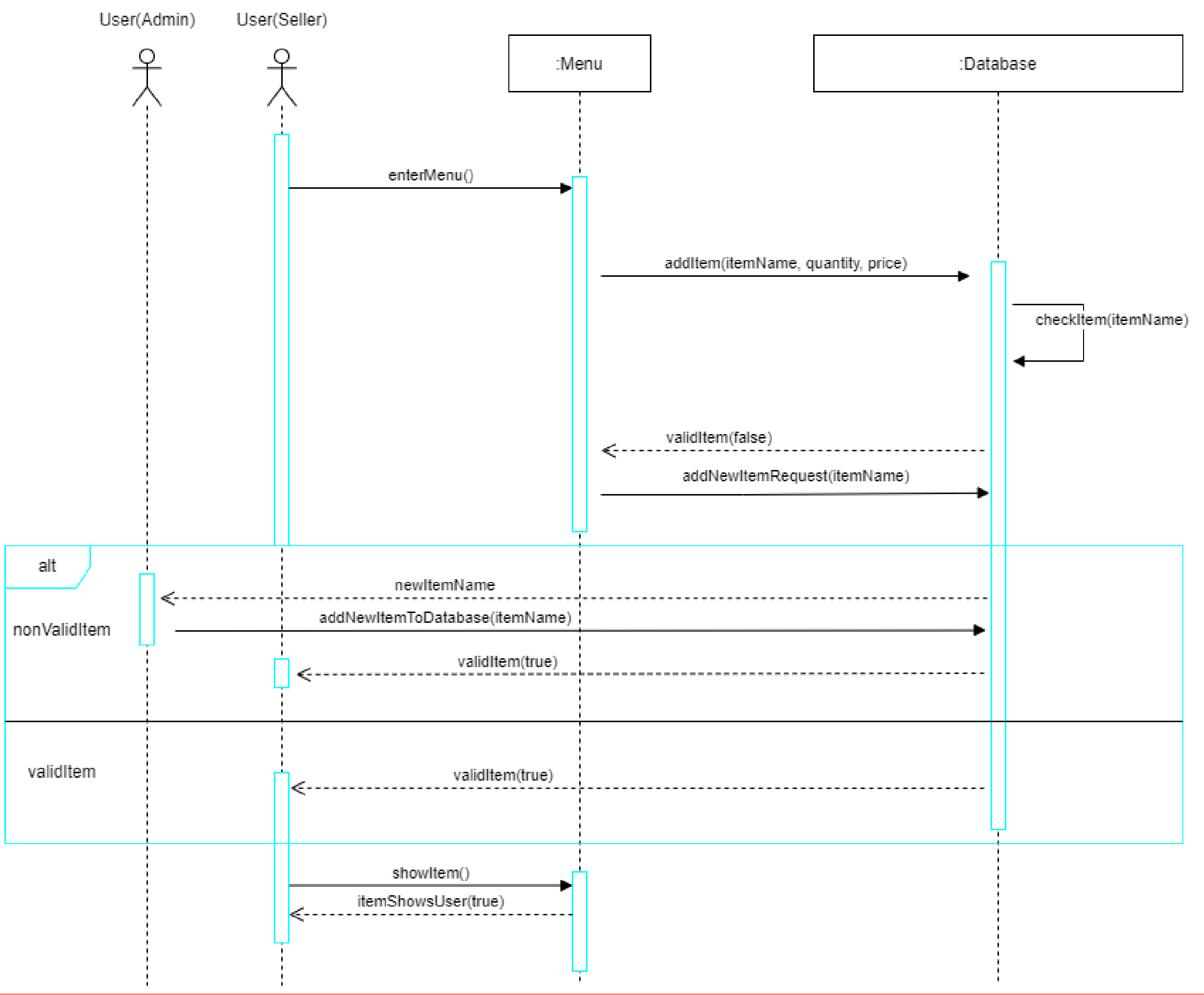
8. Class Diagram

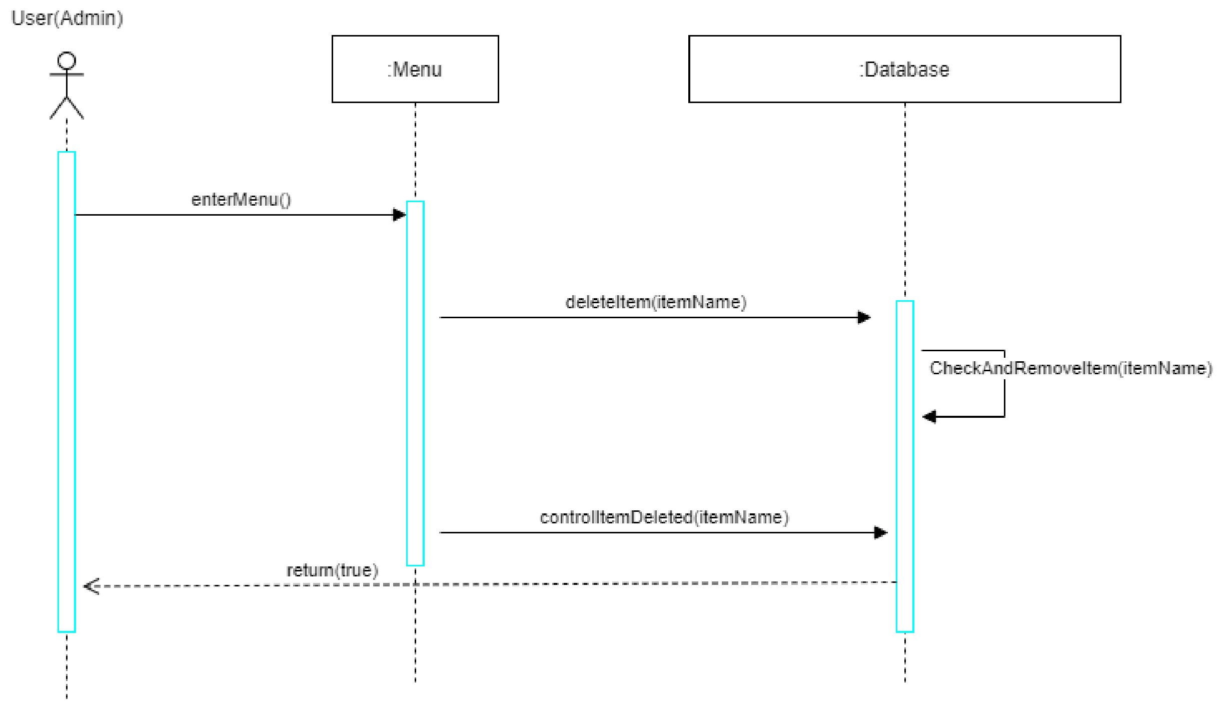


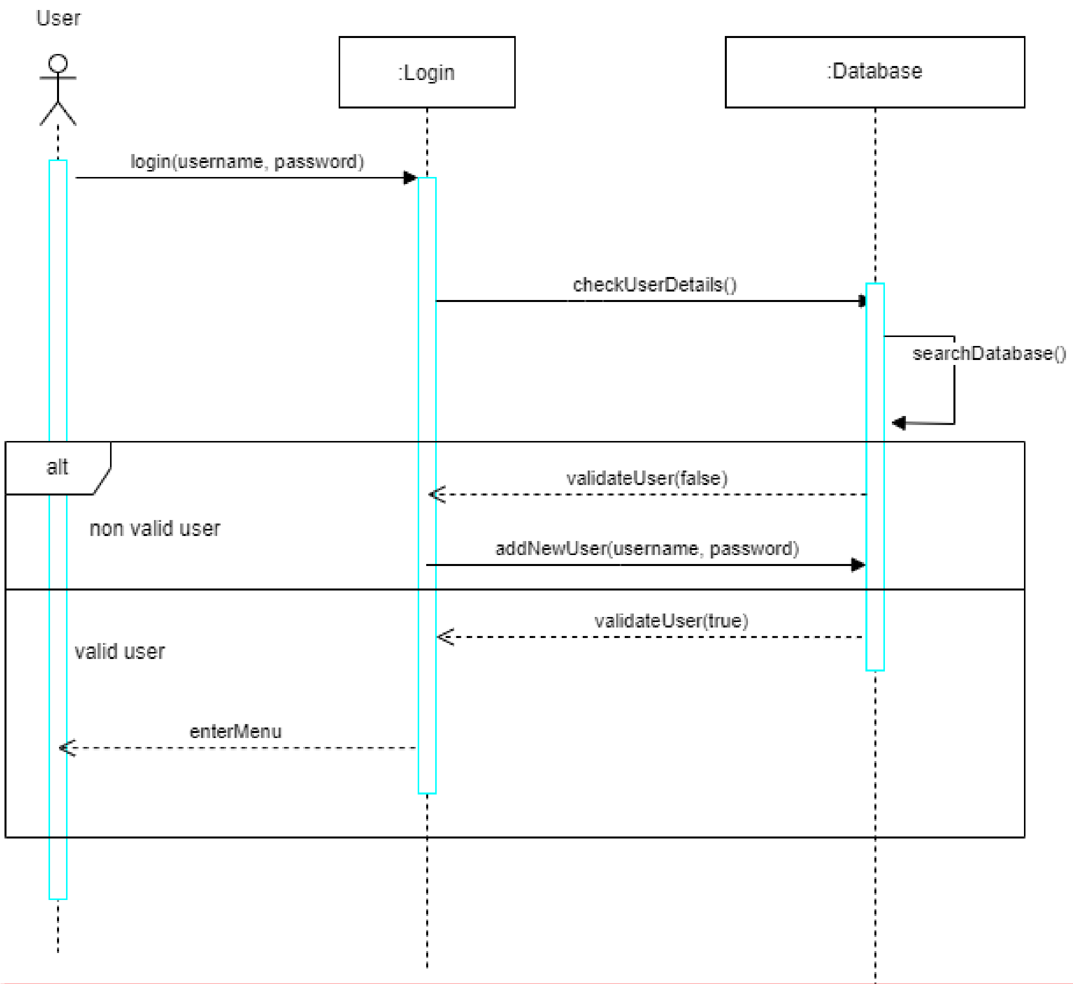
ECommerceSystem
resourcesDir : String
Sellers : Map <String, String>
Customers : Map <String, String>
Admins : Map <String, String>
products : Map<String, LinkedList<Product>>
Requests : PriorityQueue<Request>
productsOrdered :ArrayList<BinarySearchTree<Product>>
UnprocessedOrders : Map<Integer, Integer>
lastID : int
ECommerceSystem()
createBST(): void
menu(): void
addProduct(String) : void
removeProduct(String) : void
saveProducts() : void
saveCustomers() : void
saveSellers() : void
saveRequests() : void
saveOrders() : void
exit() : void
«interface» Comparator
«interface» Iterator
Driver
main (String[]): void
«interface» Comparable
Edge
source : int
dest : int
weight : double
Edge (int, int)
Edge (int, int, double)
getSource() : int
getDest() : int
getWeight() : double
toString() : String
equals(Object) : boolean
hashCode() : int
compare(Edge, Edge) : int
MatrixGraph
edges: double[][]
numV : int
directed : boolean
MatrixGraph (int, boolean)
insert (Edge): void
isEdge (int, int) : boolean
getEdge (int, int) : Edge
edgeIterator (int) : Iterator <Edge>
setEdgeValue (int, int, double) : void
getEdgeValue (int, int) : double
getNumV() : int
isDirected() : boolean
Iter
source : int
index : int
Iter (int)
hasNext() : boolean
next() : Edge
remove() : void
advanceIndex() : void
Request
priority : int
compareTo(Request) : int
SellerRequest
user : User
password : String
SellerRequest(User, String)
toString(): String
ProductRequest
productName : String
ProductRequest(String)
toString(): String
Product
productName: String
sellerName : String
price : double
stock : int
Product (String, String, double, int)
toString() : String
compareTo (Product) : int
getProductName() : String
getStock(): int
setStock(int) : void
getSellerName() : String
getPrice() : double
clone() : Product
User
systemRef : ECommerceSystem
username : String
User (String, ECommerceSystem)
UI(): void
getProduct (String, String) : Product
getProduct (String) : LinkedList <Product>
isInteger(String) : boolean
incrementID(): void
getProductsMap () : Map<String, LinkedList<Product>>
getID () : int
updateOrders (int, int) : void
getOrders () : HashMap<Integer, Integer>
getProducts() : ArrayList<BinarySearchTree<Product>>
updateBST (): void
changePass(String) : void
getInputInt (Scanner, String) : int
newProductRequest(String): boolean
Admin
Admin (String, ECommerceSystem)
UI () : void
«interface» SearchTree<E>
«interface» Serializable
BinaryTree <E>
root : Node <E>
BinaryTree()
BinaryTree(Node <E>)
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
getLeftSubtree(): BinaryTree<E>
getRightSubtree(): BinaryTree<E>
getData() : E
isLeaf(): boolean
toString() : String
traverse(Node<E>, int, StringBuilder): void
readBinaryTree (Scanner) : BinaryTree<String>
BinarySearchTree <E>
adder : boolean
deleter : E
BinarySearchTree()
BinarySearchTree(Node<E>)
add(E) : boolean
add(Node<E>,E) : Node<E>
contains (E) : boolean
find(E) : E
find(Node<E>, E) : E
delete(E) : E
delete(Node<E>,E) : Node<E>
remove(E) : boolean
getLeftSubtree () : BinarySearchTree <E>
getRightSubtree () : BinarySearchTree <E>
largestChild(Node<E>) : E
Node <E>
data : E
left : Node <E>
right : Node <E>
Node(E)
toString(): String
SkipList <E>
head: SLNode<E>
maxLevel : int
maxCap : int
size : int
LOG2 : double
rand : Random
SkipList()
logRandom() : int
insert (E) : void
computeMaxCap (int) : int
search (E) : SLNode <E> []
find (E) : E
isEmpty () : boolean
toArray() : E[]
SLNode<E>
links : SLNode<E> []
data : E
SLNode (int , E)
Seller
orderHistory: SkipList <Order>
waitingOrders : Queue <Order>
prdouctList : ArrayList <Product>
Seller (String, ECommerceSystem)
UI() : void
saveToFile() : void
productAvailable (String) : Product
addOrder (Product, int, int, String, String, String) : void
Customer
basket : LinkedList<Pair<Product, Integer>>
formerOrders: LinkedList<Pair<Pair<Product, Integer>, Pair<Integer, Integer>>>
orders: LinkedList<Pair<Pair<Product, Integer>, Pair<Integer, Integer>>>
graph : MatrixGraph
wallet : double
Customer (String, ECommerceSystem)
exit(): void
UI(): void
ascendingTraversal (BinarySearchTree<Product>) : void
descendingTraversal : (BinarySearchTree<Product>) : void
displayProduct(): void
quickSort(ArrayList<String>): void
recursiveQuickSort (ArrayList <String> , int, int) : void
partition (ArrayList <String> , int, int) : int
swap(ArrayList <String> , int, int) : void
Order
customer : String
address : String
phoneNum : String
product : Product
ID : int
quantity : int
Order (String)
Order (Product, int, int, String, String, String)
accept() : void
reject() : void
setQuantity (int) : boolean
toString(): String
compareTo (Order) : int
Pair <K,V>
key : K
value : V
Pair ()
Pair (K,V)
getKey () : K
getValue () : V

You can access high resolution image from : ClassDiagram.jpg

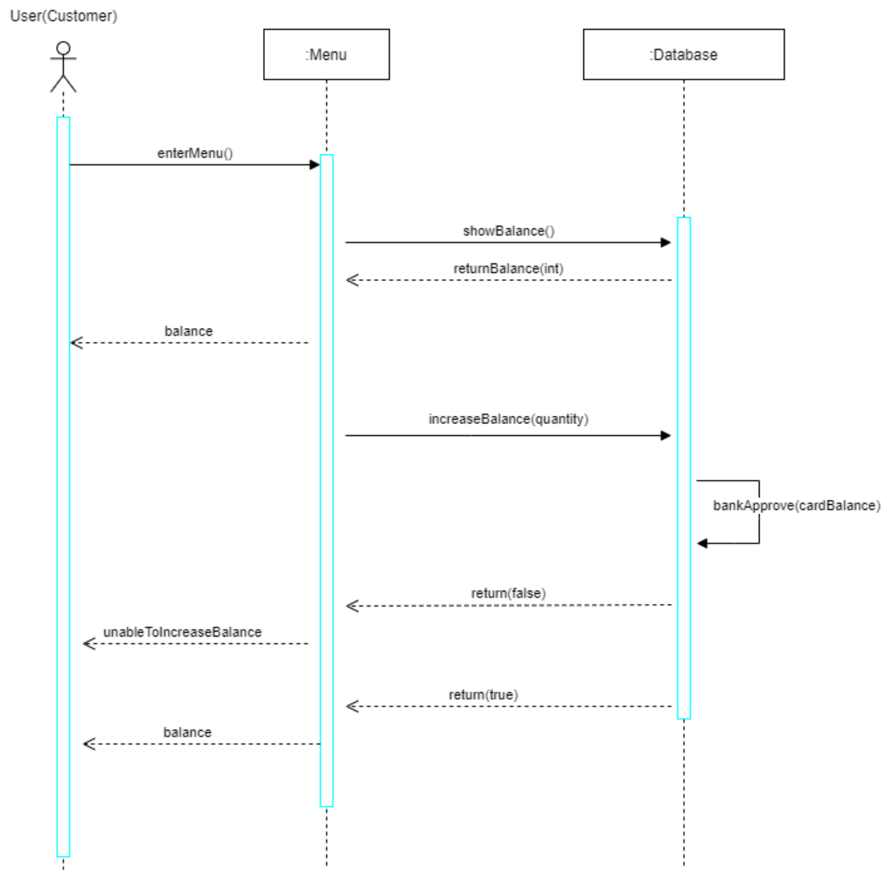
9. Sequence Diagrams

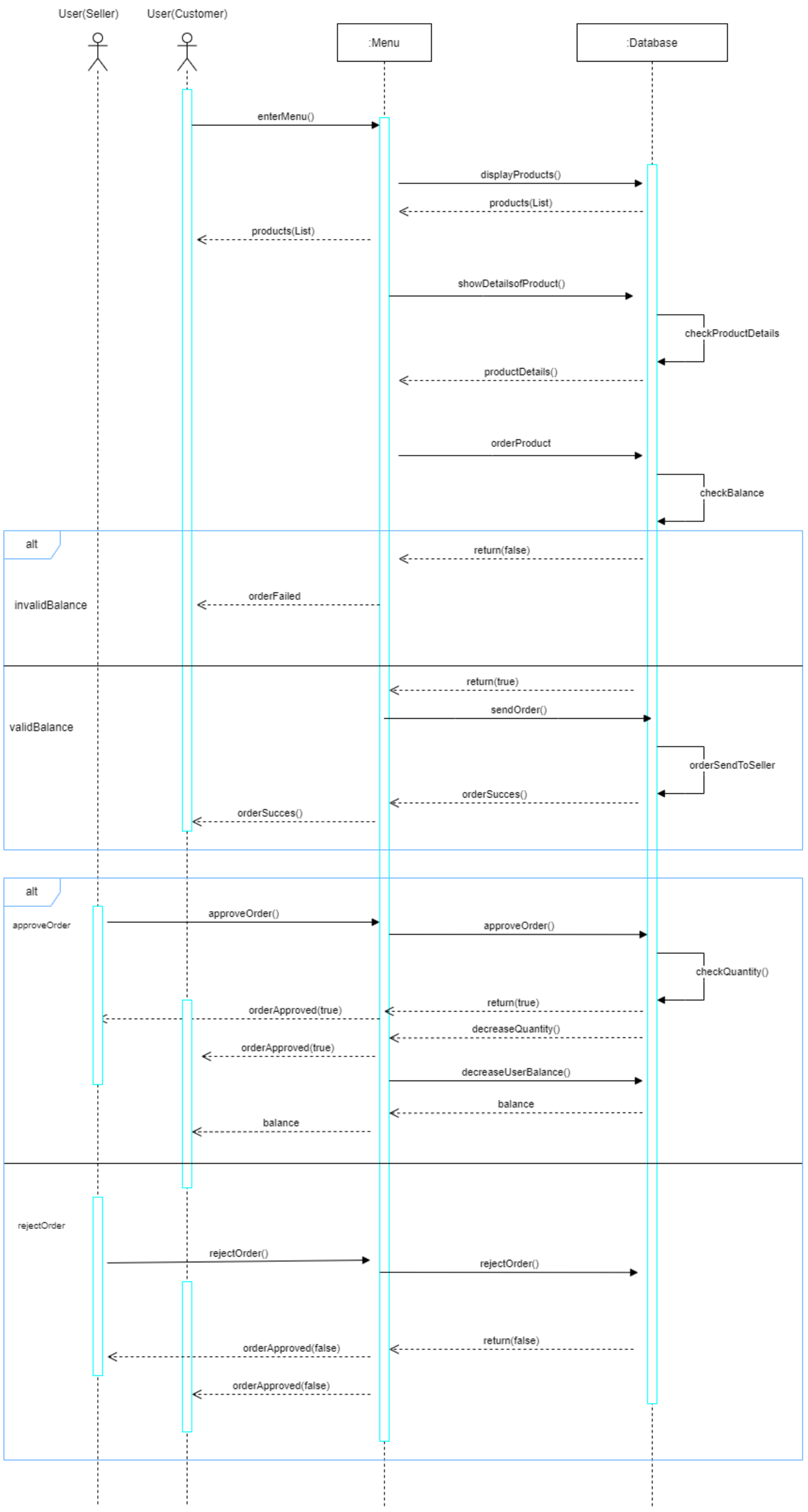

User(Admin)
User(Seller)
:Menu
:Database
enterMenu()
addItem(itemName, quantity, price)
checkItem(itemName)
validItem(false)
addNewItemRequest(itemName)
alt
nonValidItem
newItemName
addNewItemToDatabase(itemName)
validItem(true)
validItem
validItem(true)
showItem()
itemShowsUser(true)


User(Admin)
:Menu
:Database
enterMenu()
deleteItem(itemName)
CheckAndRemoveItem(itemName)
controlItemDeleted(itemName)
return(true)

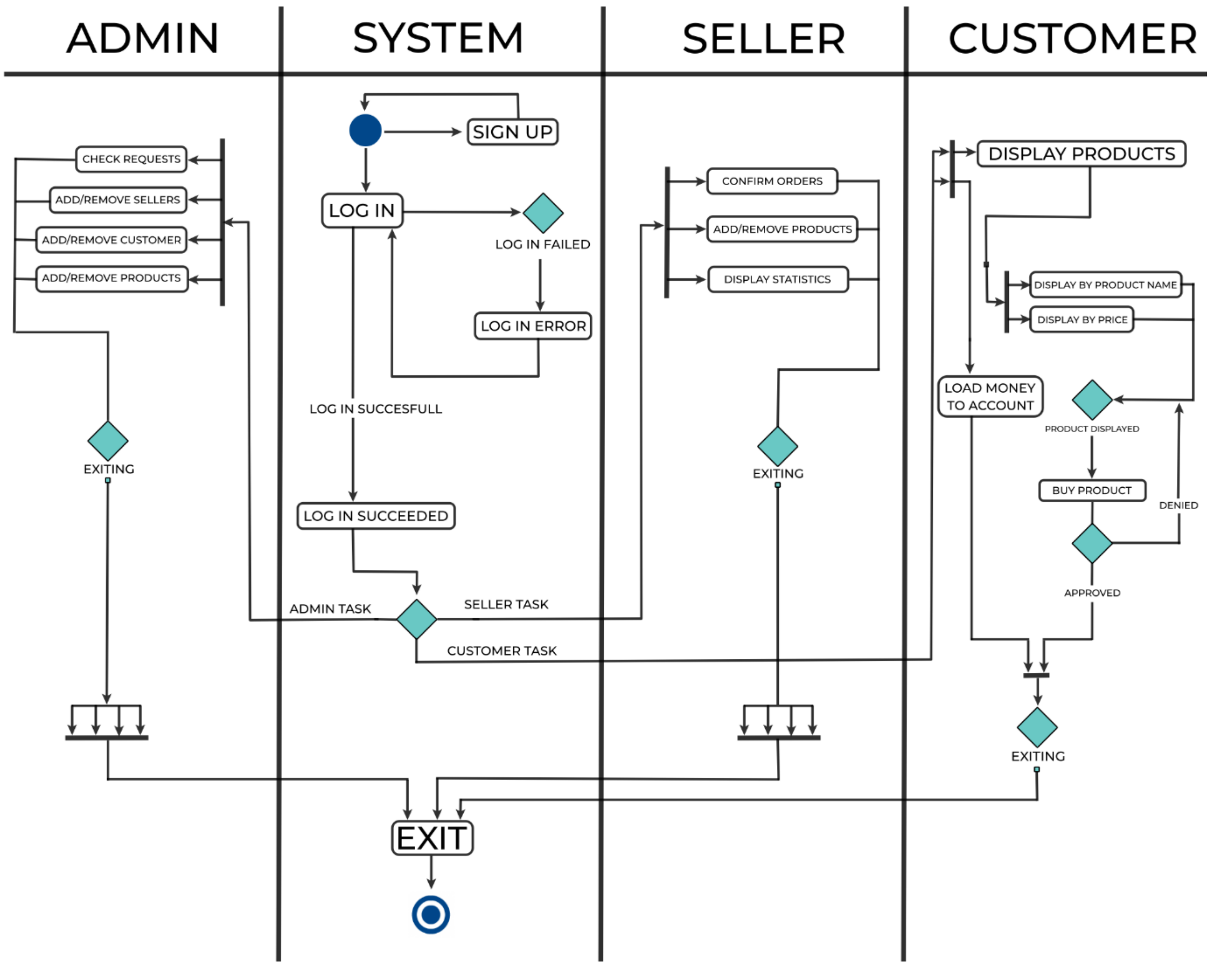


Wallet Sequence




User(Seller)
User(Customer)
:Menu
:Database
enterMenu()
displayProducts()
products(List)
products(List)
showDetailsofProduct()
checkProductDetails
productDetails()
orderProduct
checkBalance
alt
return(false)
invalidBalance
orderFailed
return(true)
sendOrder()
validBalance
orderSendToSeller
orderSucces()
orderSucces()
alt
approveOrder()
approveOrder
approveOrder()
checkQuantity()
return(true)
orderApproved(true)
decreaseQuantity()
orderApproved(true)
decreaseUserBalance()
balance
balance
rejectOrder
rejectOrder()
rejectOrder()
return(false)
orderApproved(false)
orderApproved(false)

10. Activity Diagram



11. Non-trivial Implementation Details

- We created and combined all users in the eCommerce system under the User abstract class.

- We use login and signup method in our system and according to these methods we separate the login and signup processes due to customer, seller or admin and enable the user to use their own features

- The Customer, the seller and Admin classes extend the User class and all classes have common features. We also created a product to check and do some operations on it and System class which controls UI and users order trace operations.

- When adding a product first the product pool is checked in the database, then the seller sends the product name, price and product quantity to the admin as a request to add this product. If the admin approves this request the product will be shown in products to all users.

- With the help of Product class products are sorted by seller or price, then the customer enters the product key into the system and the order request is sent to the seller with the help of sendRequest() function in Customer class. In seller class the seller checks this order from the database. Approves or rejects

- Customers have their wallets in the system and can use the created wallet inside the Wallet class which is the inner class of Customer Class. Users can increase their balance (increase ()), make payment(makePayment()) from their balance.

Which data structure we use in which class?

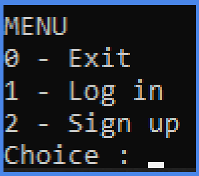
- We use HashMap for Sellers, Customers and Admins. We use HashMap data structure because everyone must have their own unique username. HashMap data structure is most convenient.

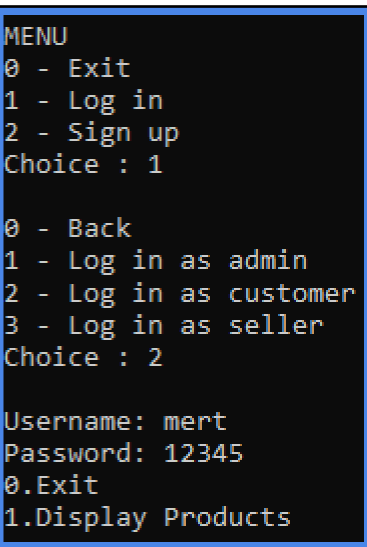
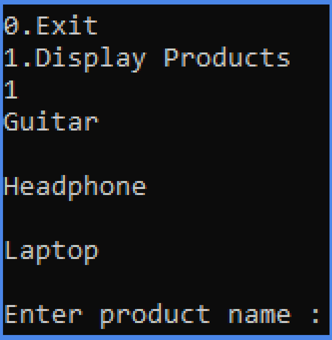
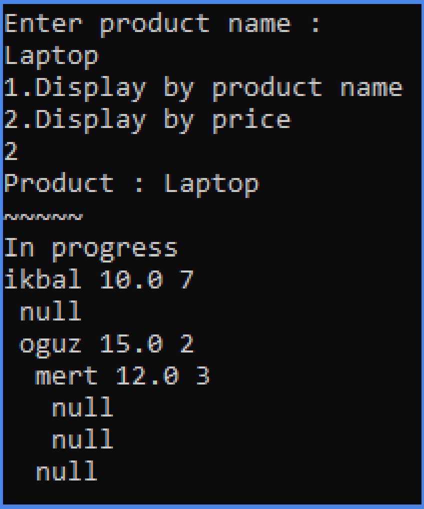
- We used Priority Queue for orders and requests. (In System class)

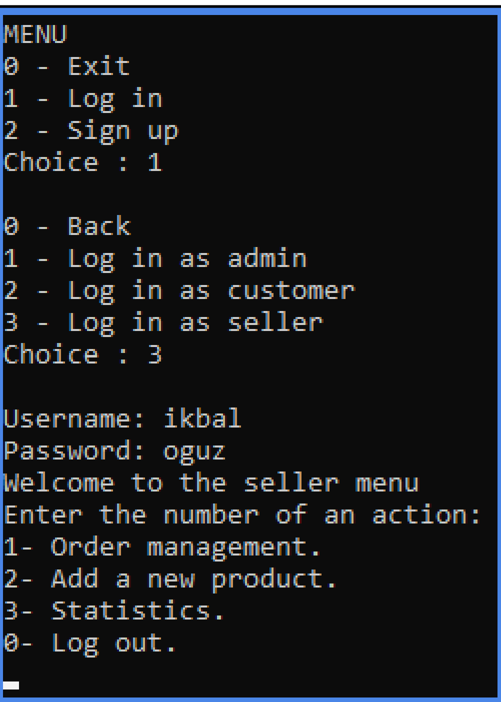
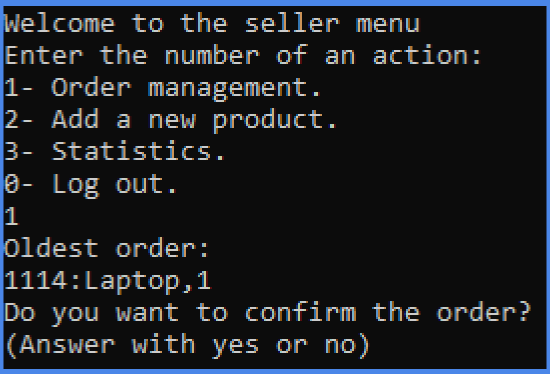
- We use LinkedList and Map data structure together to keep products. The reason we keep these two data structures together is because we need access to be able to list products seller and create orders (In product class)

- We use Binary Search Tree and ArrayList for ordering and listing the products. The reason we use these two data structures is that we can access products in a shorter time with Binary Search Tree while sorting from ascending order, and the reason why we use Arraylist for products is because it is easy to implement. Every index of ArrayList is a Binary Search Tree each product can easily sort by price and list.

12. Test Cases

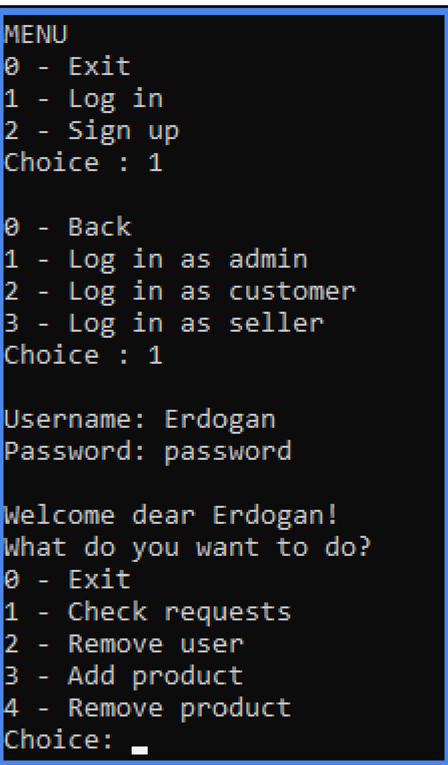
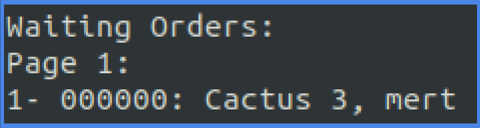
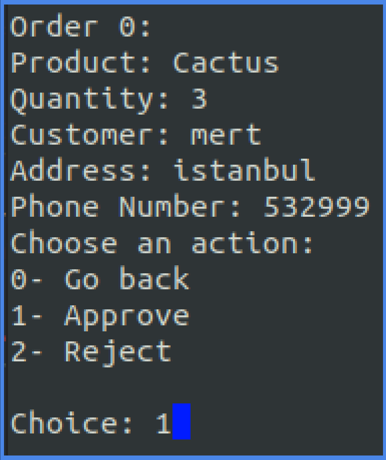
Unit Tests	How	Result
Is E-Commerce system menu working?	 ```MENU``` ```0 - Exit``` ```1 - Log in``` ```2 - Sign up``` ```Choice : _```	Menu showed

Customer login	 MENU 0 - Exit 1 - Log in 2 - Sign up Choice : 1 0 - Back 1 - Log in as admin 2 - Log in as customer 3 - Log in as seller Choice : 2 Username: mert Password: 12345 0.Exit 1.Display Products	Logged in
Customer displaying products	 0.Exit 1.Display Products 1 Guitar Headphone Laptop Enter product name :	Products displayed
List product sellers by price	 Enter product name : Laptop 1.Display by product name 2.Display by price 2 Product : Laptop ~~~~~~ In progress ikbal 10.0 7 null oguz 15.0 2 mert 12.0 3 null null null	Listed by price

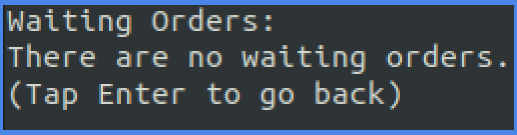
Seller login		Logged in
Show waiting orders		The oldest order is shown

```
MENU
0 - Exit
1 - Log in
2 - Sign up
Choice : 1

0 - Back
1 - Log in as admin
2 - Log in as customer
3 - Log in as seller
Choice : 3

Username: ikbal
Password: oguz
Welcome to the seller menu
Enter the number of an action:
1- Order management.
2- Add a new product.
3- Statistics.
0- Log out.
```

```
Welcome to the seller menu
Enter the number of an action:
1- Order management.
2- Add a new product.
3- Statistics.
0- Log out.
1
Oldest order:
1114:Laptop,1
Do you want to confirm the order?
(Answer with yes or no)
```

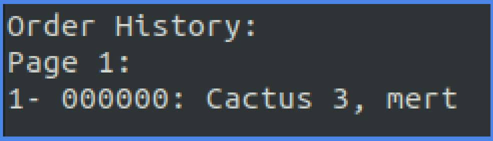
Admin login	 MENU 0 - Exit 1 - Log in 2 - Sign up Choice : 1 0 - Back 1 - Log in as admin 2 - Log in as customer 3 - Log in as seller Choice : 1 Username: Erdogan Password: password Welcome dear Erdogan! What do you want to do? 0 - Exit 1 - Check requests 2 - Remove user 3 - Add product 4 - Remove product Choice:	Logged in
Is the password system working?	As tested above in the sign in tests.	OK
Are class seller methods working?	Seeing waiting orders:  Waiting Orders: Page 1: 1- 000000: Cactus 3, mert Examining the order:  Order 0: Product: Cactus Quantity: 3 Customer: mert Address: istanbul Phone Number: 532999 Choose an action: 0- Go back 1- Approve 2- Reject Choice: 1	OK

Empty orders:



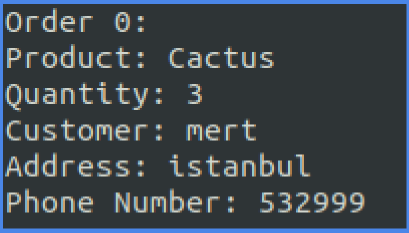
```
Waiting Orders:
There are no waiting orders.
(Tap Enter to go back)
```

Seeing order history:



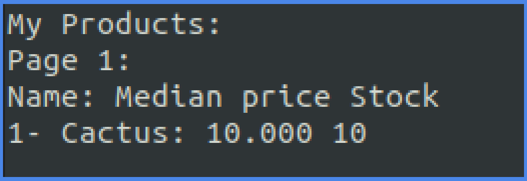
```
Order History:
Page 1:
1- 000000: Cactus 3, mert
```

Examining the previous order:



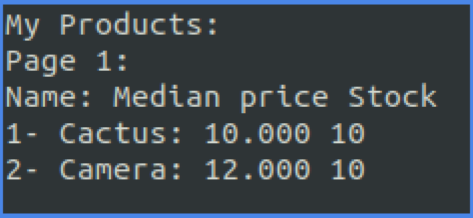
```
Order 0:
Product: Cactus
Quantity: 3
Customer: mert
Address: istanbul
Phone Number: 532999
```

Viewing the products:

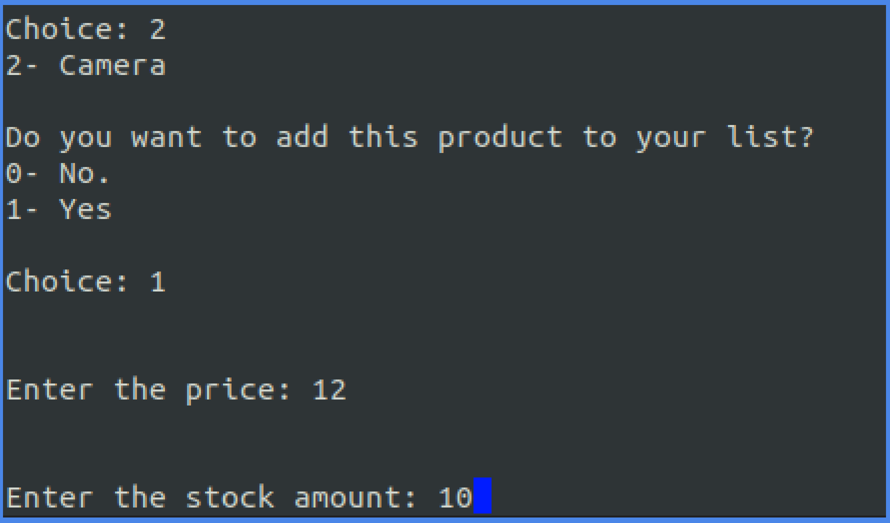
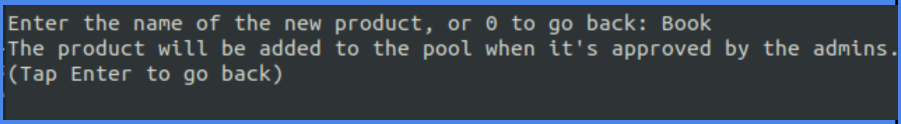
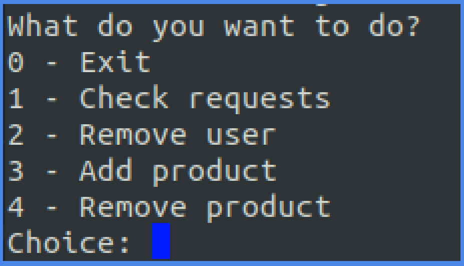
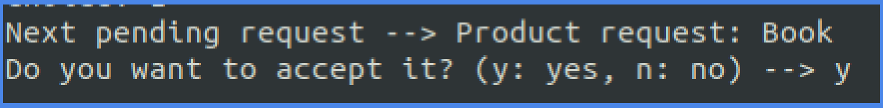
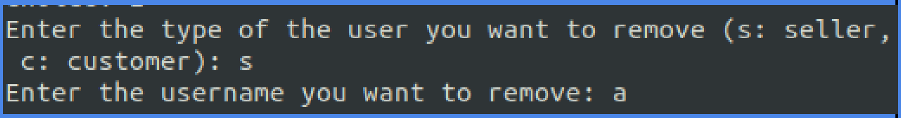


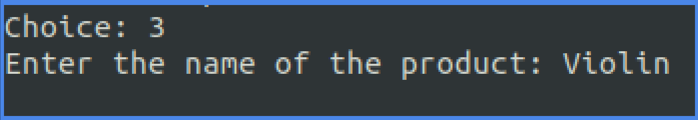
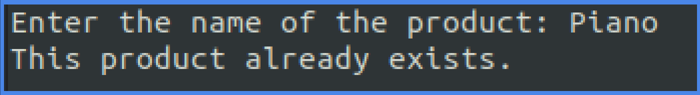
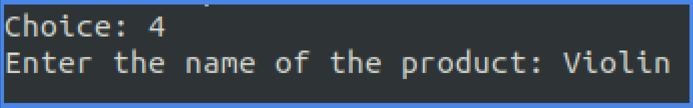
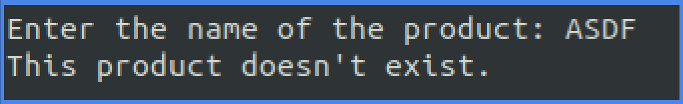
```
My Products:
Page 1:
Name: Median price Stock
1- Cactus: 10.000 10
```

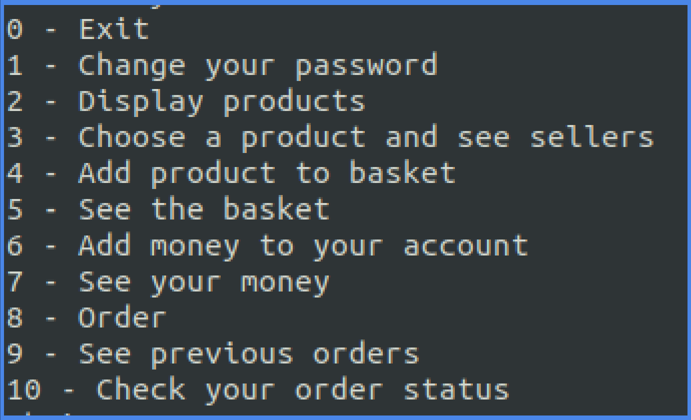
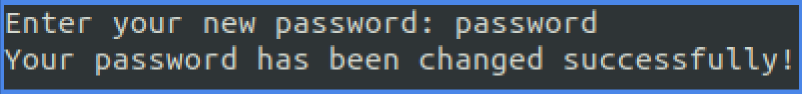
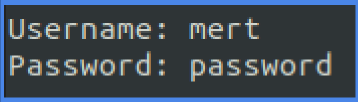
Adding product to the list:



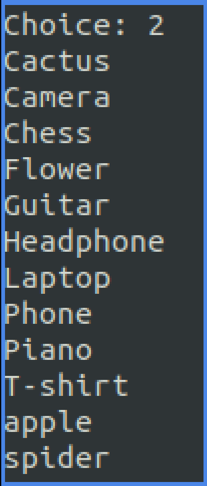
```
My Products:
Page 1:
Name: Median price Stock
1- Cactus: 10.000 10
2- Camera: 12.000 10
```

	Product request:  `Choice: 2` `2- Camera` `Do you want to add this product to your list?` `0- No.` `1- Yes` `Choice: 1` `Enter the price: 12` `Enter the stock amount: 10`  `Enter the name of the new product, or 0 to go back: Book` `The product will be added to the pool when it's approved by the admins.` `(Tap Enter to go back)`	
Are class order methods working?	All the methods and constructors are working properly based on seller class operations.	All working
Are class admin methods working?	The admin class's UI method and all its operations are functioning properly. Choices:  `What do you want to do?` `0 - Exit` `1 - Check requests` `2 - Remove user` `3 - Add product` `4 - Remove product` `Choice:` Checking requests:  `Next pending request --> Product request: Book` `Do you want to accept it? (y: yes, n: no) --> y` Removing user:  `Enter the type of the user you want to remove (s: seller, c: customer): s` `Enter the username you want to remove: a`	All working

	Adding product:
	
	`Choice: 3`
	`Enter the name of the product: Violin`
	
	`Enter the name of the product: Piano`
	`This product already exists.`
	Removing product:
	
	`Choice: 4`
	`Enter the name of the product: Violin`
	
	`Enter the name of the product: ASDF`
	`This product doesn't exist.`	

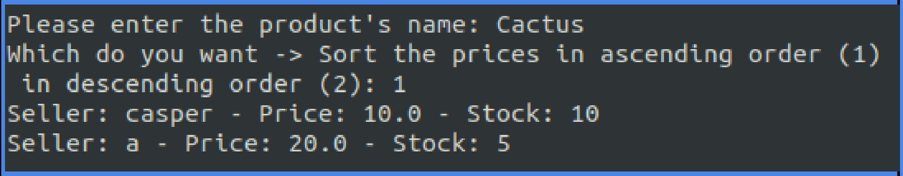
Are class customer methods working?	Choices:
	
	`0 - Exit`
	`1 - Change your password`
	`2 - Display products`
	`3 - Choose a product and see sellers`
	`4 - Add product to basket`
	`5 - See the basket`
	`6 - Add money to your account`
	`7 - See your money`
	`8 - Order`
	`9 - See previous orders`
	`10 - Check your order status`
	Changing password:
	
	`Enter your new password: password`
	`Your password has been changed successfully!`
	
	`Username: mert`
	`Password: password`
		OK

Displaying products:



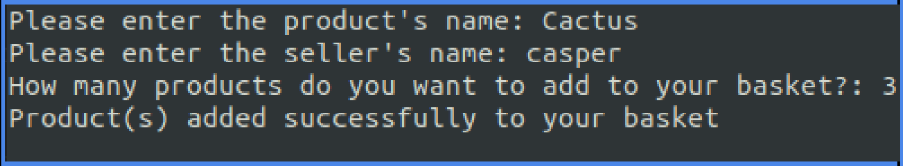
```
Choice: 2
Cactus
Camera
Chess
Flower
Guitar
Headphone
Laptop
Phone
Piano
T-shirt
apple
spider
```

Choosing a product and see sellers:



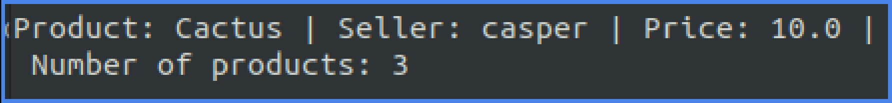
```
Please enter the product's name: Cactus
Which do you want -> Sort the prices in ascending order (1)
 in descending order (2): 1
Seller: casper - Price: 10.0 - Stock: 10
Seller: a - Price: 20.0 - Stock: 5
```

Adding product to basket:



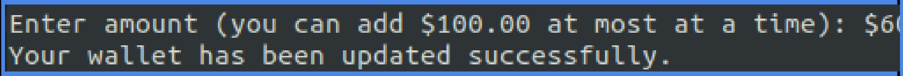
```
Please enter the product's name: Cactus
Please enter the seller's name: casper
How many products do you want to add to your basket?: 3
Product(s) added successfully to your basket
```

Seeing the basket:



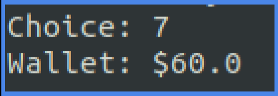
```
Product: Cactus | Seller: casper | Price: 10.0 |
 Number of products: 3
```

Adding money to account:



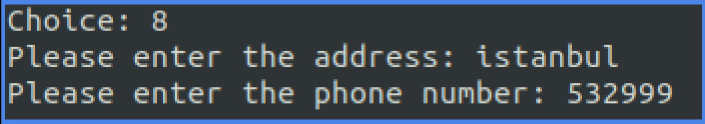
```
Enter amount (you can add $100.00 at most at a time): $6
Your wallet has been updated successfully.
```

Seeing the money:

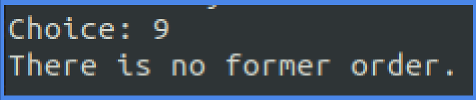
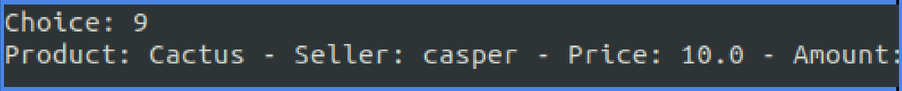
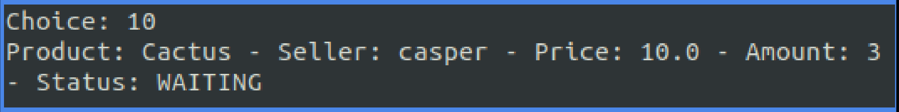
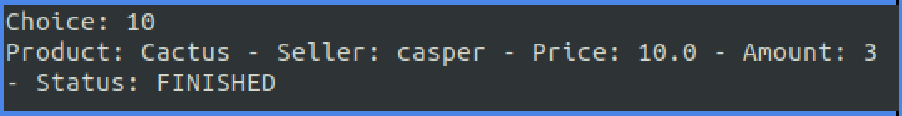


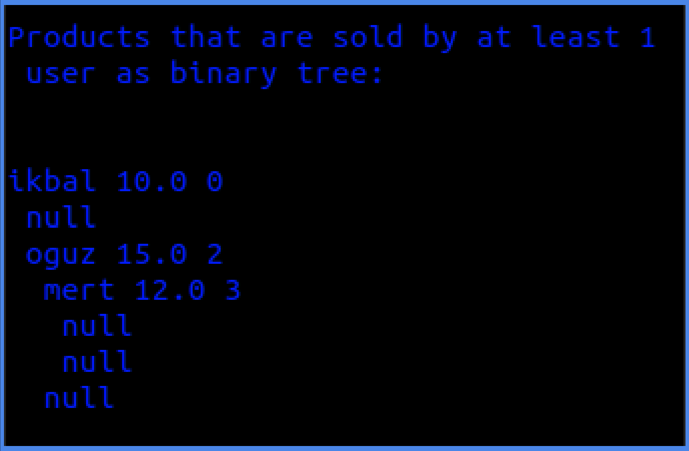
```
Choice: 7
Wallet: $60.0
```

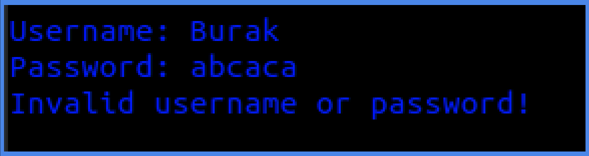
Ordering:

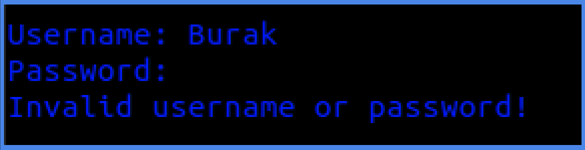


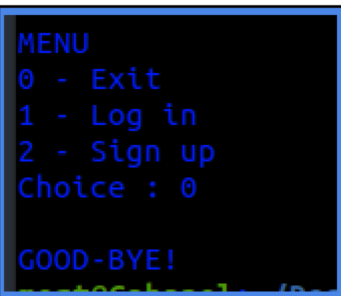
```
Choice: 8
Please enter the address: istanbul
Please enter the phone number: 532999
```

	Seeing previous orders:
	
	`Choice: 9`
	`There is no former order.`
	
	`Choice: 9`
	`Product: Cactus - Seller: casper - Price: 10.0 - Amount:`
	
	Checking the order status:
	
	`Choice: 10`
	`Product: Cactus - Seller: casper - Price: 10.0 - Amount: 3`
	`- Status: WAITING`
	
	`Choice: 10`
	`Product: Cactus - Seller: casper - Price: 10.0 - Amount: 3`
	`- Status: FINISHED`	

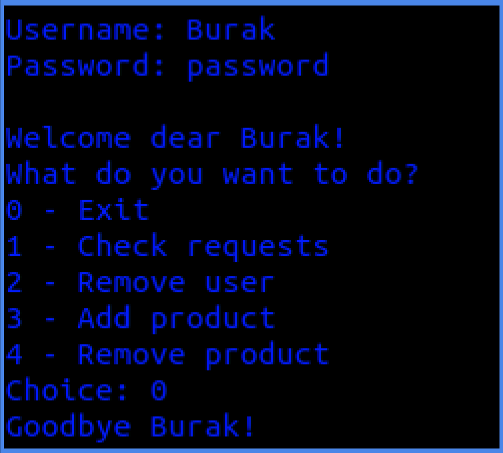
Is binary tree working truly?	
	`Products that are sold by at least 1`
	` user as binary tree:`
	
	`ikbal 10.0 0`
	` null`
	` oguz 15.0 2`
	`  mert 12.0 3`
	`   null`
	`   null`
	`  null`
		OK

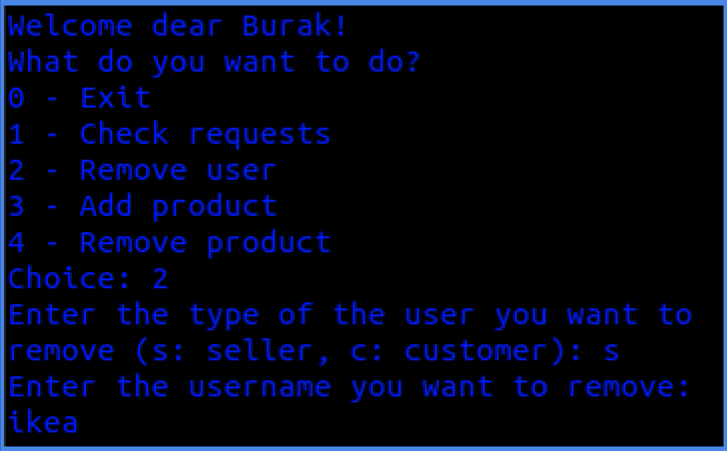
Does password check letters?	Ex: “abcaca”
	
	`Username: Burak`
	`Password: abcaca`
	`Invalid username or password!`
		OK

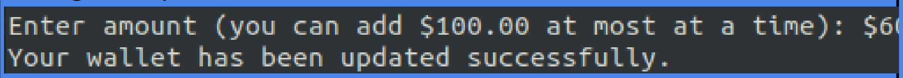
Does password check empty string?	Ex: “ “
	
	`Username: Burak`
	`Password:`
	`Invalid username or password!`
		OK

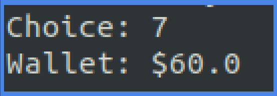
Can system exit safely?	
	`MENU`
	`0 - Exit`
	`1 - Log in`
	`2 - Sign up`
	`Choice : 0`
	
	`GOOD-BYE!`
		OK

Can system record information to files in any unexpected exit?	Exit forcedly Thanks to exit() method used in menu, files can be updated properly.	OK

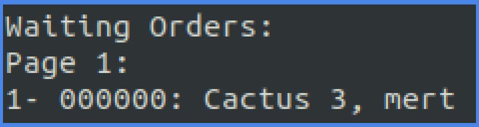
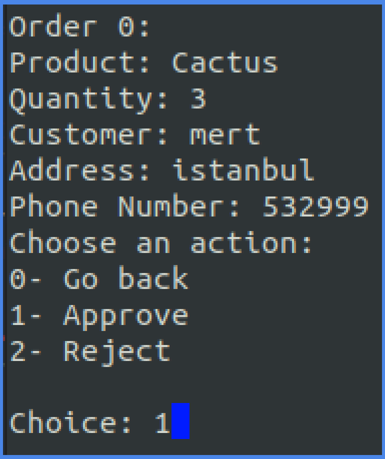
Can users sign in and out?	Exit normally
	
	
	`Username: Burak`
	`Password: password`
	
	`Welcome dear Burak!`
	`What do you want to do?`
	`0 - Exit`
	`1 - Check requests`
	`2 - Remove user`
	`3 - Add product`
	`4 - Remove product`
	`Choice: 0`
	`Goodbye Burak!`
		OK

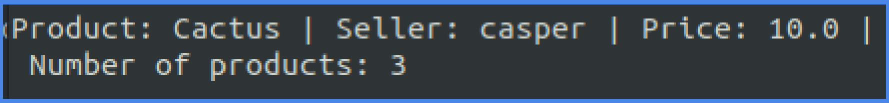
Can admin delete a user?	
	`Welcome dear Burak!`
	`What do you want to do?`
	`0 - Exit`
	`1 - Check requests`
	`2 - Remove user`
	`3 - Add product`
	`4 - Remove product`
	`Choice: 2`
	`Enter the type of the user you want to remove (s: seller, c: customer): s`
	`Enter the username you want to remove: ikea`
		OK

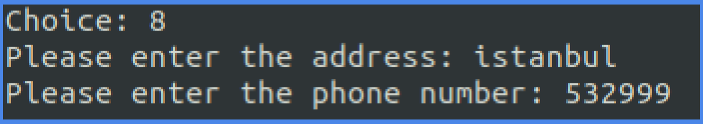
Does the customer wallet work properly?	Customer class will be used.
	
	Adding money to account:
	
	`Enter amount (you can add $100.00 at most at a time): $6`
	`Your wallet has been updated successfully.`
		OK

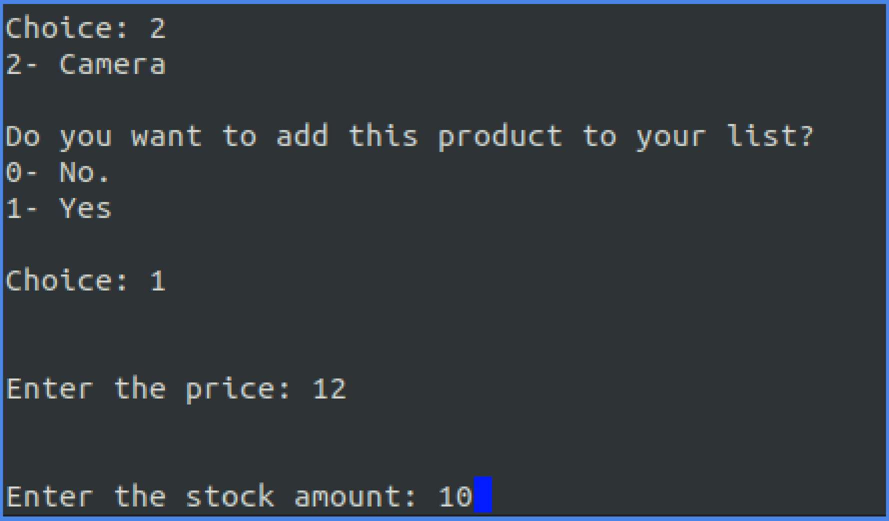
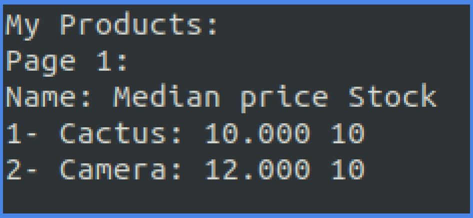
	Seeing the money:
	
	`Choice: 7`
	`Wallet: $60.0`	

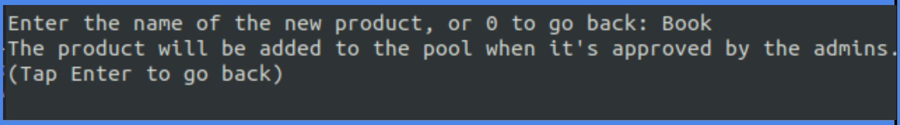
Integration Tests	How	Result

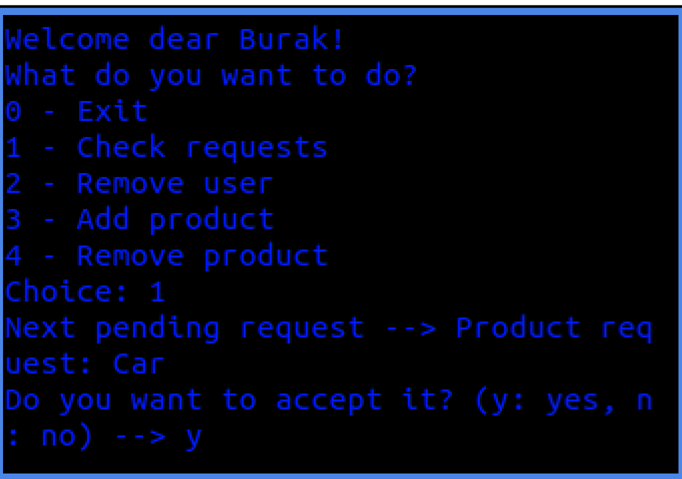
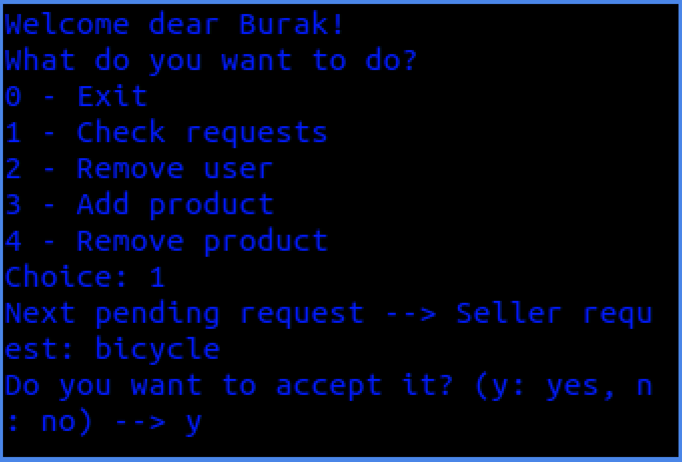
Approving order	Seller class will be used.
	
	Seeing waiting orders:
	
	`Waiting Orders:`
	`Page 1:`
	`1- 000000: Cactus 3, mert`
	
	Examining the order:
	
	`Order 0:`
	`Product: Cactus`
	`Quantity: 3`
	`Customer: mert`
	`Address: istanbul`
	`Phone Number: 532999`
	`Choose an action:`
	`0- Go back`
	`1- Approve`
	`2- Reject`
	
	`Choice: 1`
		OK

Creating order	Customer class will be used.
	
	Basket:
	
	`Product: Cactus | Seller: casper | Price: 10.0 |`
	` Number of products: 3`
		OK

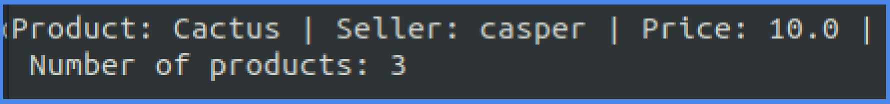
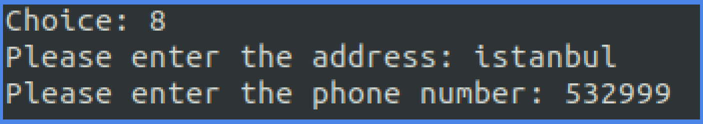
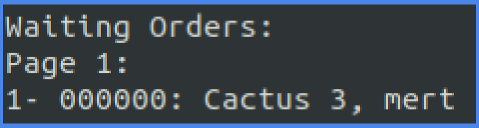
	Order:
	
	`Choice: 8`
	`Please enter the address: istanbul`
	`Please enter the phone number: 532999`	

Seller adding product	Seller class will be used.
	
	Adding product to the list:
	
	`Choice: 2`
	`2- Camera`
	
	`Do you want to add this product to your list?`
	`0- No.`
	`1- Yes`
	
	`Choice: 1`
	
	`Enter the price: 12`
	
	`Enter the stock amount: 10`
	
	`My Products:`
	`Page 1:`
	`Name: Median price Stock`
	`1- Cactus: 10.000 10`
	`2- Camera: 12.000 10`
		OK

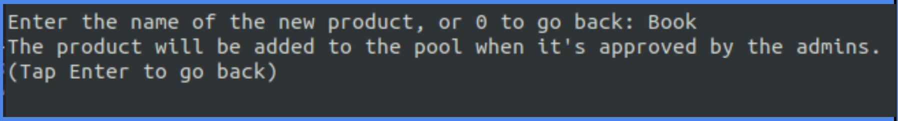
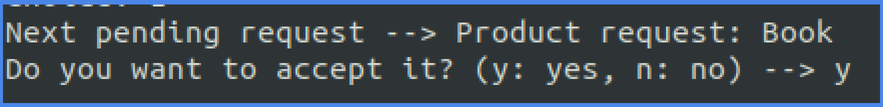
New product request	Seller class will be used.
	
	Product request:
	
	`Enter the name of the new product, or 0 to go back: Book`
	`The product will be added to the pool when it's approved by the admins.`
	`(Tap Enter to go back)`
		OK

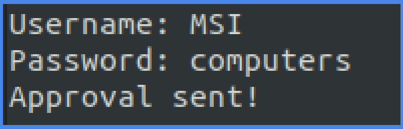
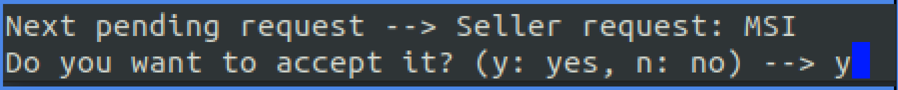
Admin approving request	
	`Welcome dear Burak!`
	`What do you want to do?`
	`0 - Exit`
	`1 - Check requests`
	`2 - Remove user`
	`3 - Add product`
	`4 - Remove product`
	`Choice: 1`
	`Next pending request --> Product req`
	`uest: Car`
	`Do you want to accept it? (y: yes, n`
	`: no) --> y`
	
	`Welcome dear Burak!`
	`What do you want to do?`
	`0 - Exit`
	`1 - Check requests`
	`2 - Remove user`
	`3 - Add product`
	`4 - Remove product`
	`Choice: 1`
	`Next pending request --> Seller requ`
	`est: bicycle`
	`Do you want to accept it? (y: yes, n`
	`: no) --> y`
		OK

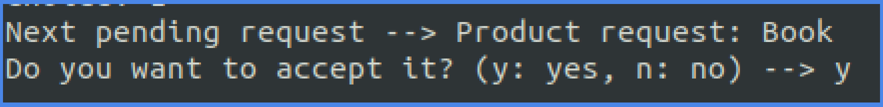
Saving to Files	Files are being saved successfully using exit() method in system.	OK

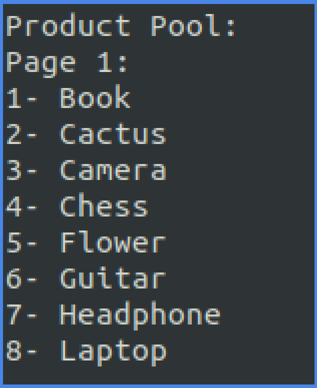
Does class seller methods communicate with class customer class truly?	Seller and Customer classes will be used.
	
	Customer sent an order:
	
	`Product: Cactus | Seller: casper | Price: 10.0 |`
	` Number of products: 3`
	
	`Choice: 8`
	`Please enter the address: istanbul`
	`Please enter the phone number: 532999`
	
	Seller can see the order:
	
	`Waiting Orders:`
	`Page 1:`
	`1- 000000: Cactus 3, mert`
		OK

Does class seller methods communicate with class	Seller and Admin classes will be used.	OK

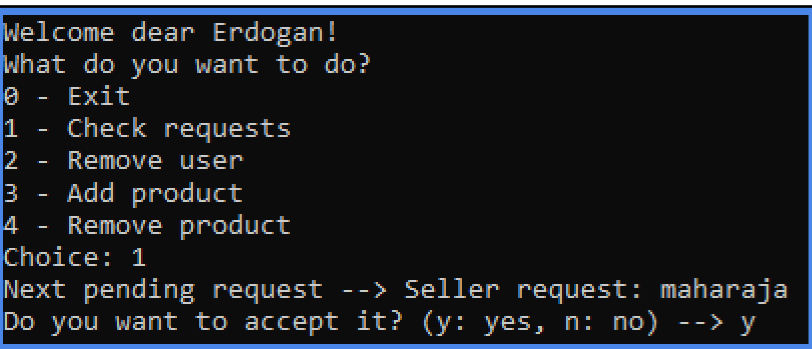
admin class truly?	Seller sent a product request:
	
	`Enter the name of the new product, or 0 to go back: Book`
	`The product will be added to the pool when it's approved by the admins.`
	`(Tap Enter to go back)`
	
	Admin can see this request:
	
	`Next pending request --> Product request: Book`
	`Do you want to accept it? (y: yes, n: no) --> y`	

Does class admin methods communicate with class seller request class truly?	Admin and SellerRequest classes will be used.
	
	Sellers can send approval as they sign up:
	
	`Username: MSI`
	`Password: computers`
	`Approval sent!`
	
	Admins can see these requests:
	
	`Next pending request --> Seller request: MSI`
	`Do you want to accept it? (y: yes, n: no) --> y`
		OK

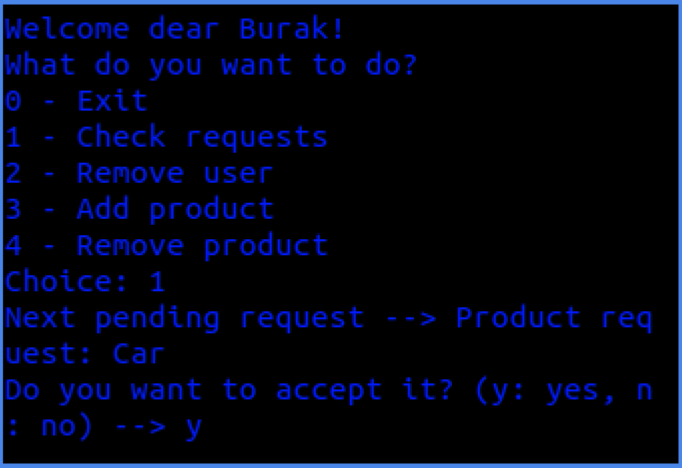
Does class admin methods communicate with class product request truly?	Admin and Product classes will be used.
	
	Admins can see the product requests:
	
	`Next pending request --> Product request: Book`
	`Do you want to accept it? (y: yes, n: no) --> y`
		OK

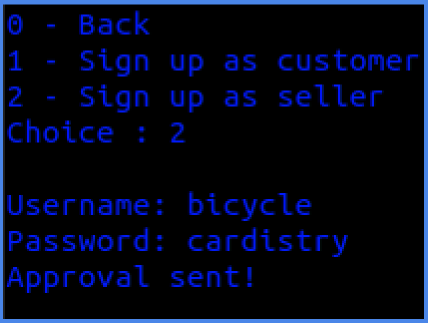
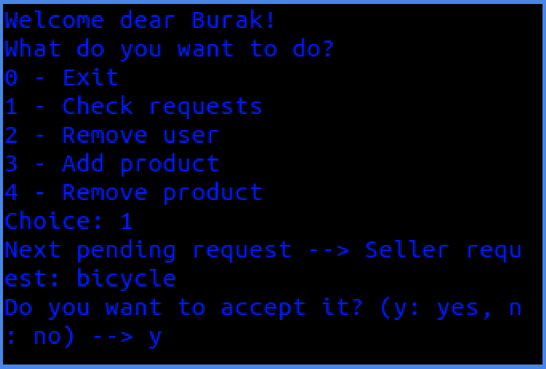
Can the class seller communicate with the system ?	Seller and ECommerceSystem classes will be used.
	
	Seller can see the products of the system while they are trying to add products to their list:
	
	`Product Pool:`
	`Page 1:`
	`1- Book`
	`2- Cactus`
	`3- Camera`
	`4- Chess`
	`5- Flower`
	`6- Guitar`
	`7- Headphone`
	`8- Laptop`
		OK

Can the class admin communicate with the system ?	Accepting sign ups in them system:	OK

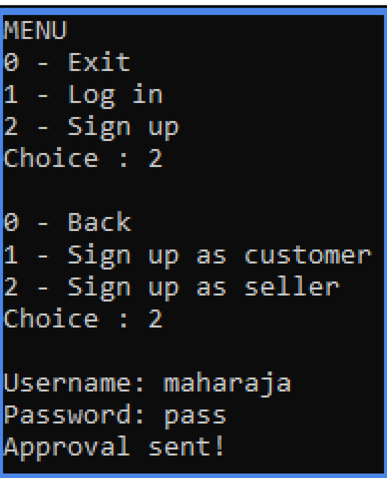
	
	```
	Welcome dear Erdogan!
	What do you want to do?
	0 - Exit
	1 - Check requests
	2 - Remove user
	3 - Add product
	4 - Remove product
	Choice: 1
	Next pending request --> Seller request: maharaja
	Do you want to accept it? (y: yes, n: no) --> y
	```

Can all class work truly?	All methods are tested.	OK

Can class admin communicate with class product request truly?	
	```
	Welcome dear Burak!
	What do you want to do?
	0 - Exit
	1 - Check requests
	2 - Remove user
	3 - Add product
	4 - Remove product
	Choice: 1
	Next pending request --> Product req
	uest: Car
	Do you want to accept it? (y: yes, n
	: no) --> y
	```
		OK

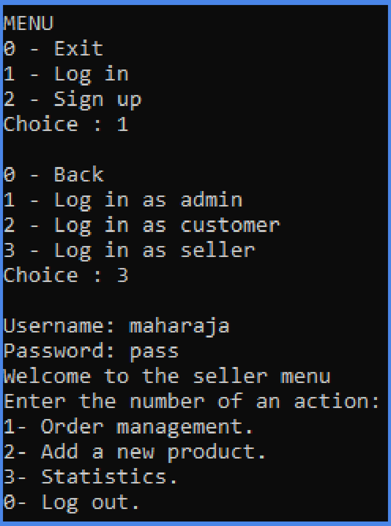
Can class admin communicate with class seller request truly?	
	```
	0 - Back
	1 - Sign up as customer
	2 - Sign up as seller
	Choice : 2

	Username: bicycle
	Password: cardistry
	Approval sent!
	```
	 
	```
	Welcome dear Burak!
	What do you want to do?
	0 - Exit
	1 - Check requests
	2 - Remove user
	3 - Add product
	4 - Remove product
	Choice: 1
	Next pending request --> Seller requ
	est: bicycle
	Do you want to accept it? (y: yes, n
	: no) --> y
	```
		OK

Can a new seller sign up in the system?	Sending sign up request:	OK



```
MENU
0 - Exit
1 - Log in
2 - Sign up
Choice : 2

0 - Back
1 - Sign up as customer
2 - Sign up as seller
Choice : 2

Username: maharaja
Password: pass
Approval sent!
```

After admin approval:



```
MENU
0 - Exit
1 - Log in
2 - Sign up
Choice : 1

0 - Back
1 - Log in as admin
2 - Log in as customer
3 - Log in as seller
Choice : 3

Username: maharaja
Password: pass
Welcome to the seller menu
Enter the number of an action:
1- Order management.
2- Add a new product.
3- Statistics.
0- Log out.
```

13. Performance Analysis

Seller Class Methods:

N = number of products in the system
n = number of products in the seller
s = number of sellers of the product
r = total number of order records

public Seller(String username, ECommerceSystem callerSystem) {	
super(username, callerSystem);	Θ(1)
orderHistory = new SkipList<>();	Θ(1)
waitingOrders = new ArrayDeque<>();	Θ(1)
productList = new ArrayList<>();	Θ(1)

```
File file = new File(systemRef.resourcesDir + "Sellers/" + username + ".txt");                    Θ(1)
if (file.exists()) {
try {
Scanner reader = new Scanner(file);       Θ(1)
String buffer = reader.nextLine();                      Θ(1)
Product targetProduct;

// Check that the file is not corrupted
// Each file must start with the name of the seller
if (!buffer.contains(username))   Θ(1)
throw new StreamCorruptedException();

// The list of products
if (reader.hasNext()) {                 Θ(1)
buffer = reader.nextLine();                       Θ(n)
String[] products = buffer.split(" ");                   Θ(n)
for (String productName : products) {     O(n)
Product target = getProduct(productName, username);   O(s*logN)
if (target != null)
productList.add(target); Θ(1)
}
}

// The list of waiting orders
if (reader.hasNext()) {
buffer = reader.nextLine();                       Θ(r)
if (!buffer.isEmpty()) {                          Θ(1)
String[] orders = buffer.split("\\|");                   Θ(r)
for (String orderString : orders)            O(r)
waitingOrders.add(new Order(orderString));       Θ(1)
}
}

// The list of past orders
if (reader.hasNext()) {                 Θ(1)
buffer = reader.nextLine();         Θ(r)
if (!buffer.isEmpty()) {                Θ(1)
String[] orders = buffer.split("\\|");                   Θ(r)
for (String orderString : orders) {           Θ(r)
Order order = new Order(orderString);   Θ(1)
orderHistory.insert(order);                   O(r)
}
}
}
} catch (FileNotFoundException e) {
System.out.print("The file was not found, default values are used.\n");               Θ(1)
} catch (NoSuchElementException | StreamCorruptedException el) {
System.out.print("The file was corrupted, default values are used.\n");               Θ(1)
file.renameTo(new File(systemRef.resourcesDir + "Sellers/" + username + "_corrupted.txt"));        Θ(1)
}
}
```

```
else
new File(systemRef.resourcesDir + "Sellers").mkdir();                 Θ(1)
}
T(N, n, r, s) = O(r^2) + O(s*n*logN) + O(r)
              = O(s*n*logN)


public void addOrder (Product product, int ID, int quantity, String customer, String phoneNum, String
address) {
waitingOrders.add(new Order(product, ID, quantity, customer, phoneNum, address));                Θ(1)
}
T(n) = Θ(1)


public void saveToFile() throws IOException {
new File(systemRef.resourcesDir + "Sellers").mkdir();                 Θ(1)
FileWriter file = new FileWriter(systemRef.resourcesDir + "Sellers/" + username + ".txt");             Θ(1)
file.write(username + "\n");                    Θ(1)

for (Product product : productList)                  Θ(n)
file.write(product.getProductName() + " ");          Θ(1)

file.write("\n");              Θ(1)

for (Order order : waitingOrders)                    Θ(r)
file.write(order + "|");              Θ(1)

file.write("\n");

Order[] old = orderHistory.toArray();        Θ(r)
if (old != null)
for (Order order : old)               Θ(r)
file.write(order + "|");    Θ(1)

file.close();
}
T(n, r) = O(r + n)



public Product productAvailable(String productName) {
for (Product product : productList)                            O(n)
if (product.getProductName().equals(productName))              Θ(1)
return product;

return null;
}
                                        T(n) = O(n)
```

Seller UI: (total complexity until operation is done)

- Showing the products of the seller: $T(n) = \Theta(n)$
- Remove a product from a seller: $T(n) = O(N*s)$

- Showing the products of the system: T(N) = Θ(N)
- Adding a new product from the pool: T(N) = O(N*s^2)
- Creating new product request for the pool: T(N) = O(N)
- Showing the orders (old and new): T(r) = Θ(r)
- Accepting or rejecting an order: T(r) = O(r)
- Navigating through pages: T(n) = Θ(1)

Customer Class Methods:

n = number of products in the system
s = number of sellers of the product
b = number of products in the basket
r = total number of order records

```
public Customer(String usernameValue, ECommerceSystem callerSystem) {
super(usernameValue, callerSystem);                Θ(1)
new File(systemRef.resourcesDir + "Customers").mkdir();                Θ(1)

try{
File file = new File(systemRef.resourcesDir + "Customers/" + username + ".txt");          Θ(1)
if (file.exists()) {                Θ(1)
Scanner scan = new Scanner(file);                Θ(1)

//Read wallet from the file
wallet = Double.parseDouble(scan.nextLine());                Θ(1)

//Read basket from the file
int basketSize = Integer.parseInt(scan.nextLine());                Θ(1)
for (int i = 0; i < basketSize; ++i){                Θ(b)
String product = scan.next();                Θ(1)
String seller = scan.next();                Θ(1)
Double price = Double.parseDouble(scan.next());                Θ(1)
int stock = Integer.parseInt(scan.next());                Θ(1)
int amount = Integer.parseInt(scan.next());                Θ(1)
basket.add(new Pair<Product, Integer>(getProduct(product, seller), amount));     O(s*logN)
}

//Read former orders from the file
int formerOrdersSize = Integer.parseInt(scan.next());                Θ(1)
for (int i = 0; i < formerOrdersSize; ++i){  Θ(r)
String product = scan.next();                Θ(1)
String seller = scan.next();                Θ(1)
Double price = Double.parseDouble(scan.next());                Θ(1)
int stock = Integer.parseInt(scan.next());                Θ(1)
int amount = Integer.parseInt(scan.next());                Θ(1)
int id = Integer.parseInt(scan.next());                Θ(1)

HashMap<Integer, Integer> orderSituations = getOrders();                Θ(1)
int situation = orderSituations.get(id);                Θ(1)
if (situation != 0){                    Θ(1)
formerOrders.add(new Pair<Pair<Product, Integer>, Pair<Integer, Integer>>(new Pair<Product,
Integer> (new Product(product, seller, price, stock), amount), new Pair<Integer, Integer>(id,
situation)));                                                                          Θ(1)
}
else
orders.add(new Pair<Pair<Product, Integer>, Pair<Integer, Integer>>(new Pair<Product, Integer>(new
Product(product, seller, price, stock), amount), new Pair<Integer, Integer>(id, situation)));
                                                                          Θ(1)
}

//Read orders from the file
int ordersSize = Integer.parseInt(scan.next());
for (int i = 0; i < ordersSize; ++i){          Θ(r)
String product = scan.next();                  Θ(1)
String seller = scan.next();                   Θ(1)
Double price = Double.parseDouble(scan.next());          Θ(1)
int stock = Integer.parseInt(scan.next());               Θ(1)
int amount = Integer.parseInt(scan.next());              Θ(1)
int id = Integer.parseInt(scan.next());                  Θ(1)

HashMap<Integer, Integer> orderSituations = getOrders();          Θ(1)
int situation = orderSituations.get(id);             Θ(1)

if (situation != 0){                    Θ(1)
formerOrders.add(new Pair<Pair<Product, Integer>, Pair<Integer, Integer>>(new Pair<Product,
Integer>(new Product(product, seller, price, stock), amount), new Pair<Integer, Integer>(id,
situation)));
if (situation == -1) wallet += amount * price;
}
Else                Θ(1)
orders.add(new Pair<Pair<Product, Integer>, Pair<Integer, Integer>>(new Pair<Product, Integer>(new
Product(product, seller, price, stock), amount), new Pair<Integer, Integer>(id, situation)));

}
}
} catch (Exception e) {
System.out.println("Error during opening the file.");
e.printStackTrace();
}

for (int i = 0; i < getID() && i < orders.size(); ++i){            O(r)
Pair<Pair<Product, Integer>, Pair<Integer, Integer>> newIndexElement = orders.get(i);             O(r)
if (newIndexElement != null && newIndexElement.getValue().getValue() != -1)
graph.insert(new Edge(newIndexElement.getValue().getKey(), i));          Θ(1)
}
}
                                                               T(b, n, r, s) = O(b*s*logn)

private void exit(){
try{
```

```
FileWriter writer = new FileWriter(systemRef.resourcesDir + "Customers/" + username + ".txt");

writer.write(wallet + "\n");

writer.write(basket.size() + "\n");
for (Pair<Product, Integer> p : basket)                Θ(b)
writer.write(p.getKey().getProductName() + " " + p.getKey() + " " + p.getValue() + "\n");

writer.write(formerOrders.size() + "\n");
for (Pair<Pair<Product, Integer>, Pair<Integer, Integer>> p : formerOrders)          Θ(r)
writer.write(p.getKey().getKey().getProductName() + " " + p.getKey().getKey() + " " +
p.getKey().getValue() + " " + p.getValue().getKey() + "\n");

writer.write(orders.size() + "\n");
for (Pair<Pair<Product, Integer>, Pair<Integer, Integer>> p : orders)                Θ(r)
writer.write(p.getKey().getKey().getProductName() + " " + p.getKey().getKey() + " " +
p.getKey().getValue() + " " + p.getValue().getKey() + "\n");

writer.close();              Θ(1)
}catch (Exception e){
e.printStackTrace();
}
}
                                                      T(b, r) = Θ(b + r)

private void displayProduct() {
Map<String, LinkedList<Product>> products = getProductsMap();
ArrayList<String> array = new ArrayList(products.size());
int i = 0;

for (Map.Entry<String, LinkedList<Product>> entry : products.entrySet()) array.add(entry.getKey());
                Θ(n)
quickSort(array);                   O(n^n) = Ω(nlogn)

for (int j = 0; j < array.size(); ++j) System.out.println(array.get(j));            Θ(n)
}
T(n) = O(n^n) = Ω(nlogn)

        (n = number of elements in the array)
public static void quickSort(ArrayList<String> array) {
recursiveQuickSort(array, 0, array.size()-1);                O(n^n) = Ω(nlogn)
}
T(n) = O(n^n) = Ω(nlogn)

private static void recursiveQuickSort(ArrayList<String> array, int first, int last) {
if (last <= first) return;
int index = partition(array, first, last);
recursiveQuickSort(array, first, index-1);            T(n/2)
recursiveQuickSort(array, index+1, last);             T(n/2)
}
                                    T(n) = O(n^n) = Ω(nlogn)
```

```
private static int partition(ArrayList<String> array, int first, int last) {
String prior = array.get(first);
int up = first, down = last;
do {
while (prior.compareTo(array.get(down)) < 0) --down;
while ((up < last) && (prior.compareTo(array.get(down)) >= 0)) ++up;
if (up < down) swap(array,up,down);
}while (down > up);
swap(array,first,down);
return down;
}
T(n) = Θ(1)
```

```
private static void swap(ArrayList<String> array, int a, int b){
String temp = array.get(a);
array.set(a, array.get(b));
array.set(b, temp);
}
T(n) = Θ(1)
```

Admin Class Methods:

n = number of products in the system
s = number of sellers of the product

```
public Admin(String usernameValue, String passwordValue) {
        super(usernameValue, passwordValue);
}
                        T(n) = Θ(1)
```

User Inner Abstract Class Methods:

n = number of products in the system
s = number of sellers of the product
r = total number of order records
S = number of sellers in the system

```
User(String usernameValue, ECommerceSystem callerSystem){
        username = usernameValue;
        systemRef = callerSystem;
}
T(n) = Θ(1)
```

```
protected Product getProduct(String productName, String seller) {
LinkedList<Product> targetList = systemRef.products.get(productName);  O(logn)
if(targetList != null) Θ(1)
```

```
for(Product temp : targetList) O(S)
if(temp.sellerName.equals(seller)) return temp; Θ(1)
return null;
}
T(n) = Θ(S)


protected LinkedList<Product> getProduct(String productName) {
return systemRef.products.get(productName); O(logn)
}
T(n) = O(logn)


public static boolean isInteger(String str) {
        if (str == null) { Θ(1)
                return false; Θ(1)
        }
        int length = str.length(); Θ(1)
        if (length == 0) { Θ(1)
                return false;
        }
        for (int i = 0; i < length; i++) { Θ(n)
                char c = str.charAt(i); Θ(1)
                if (c < '0' || c > '9') { Θ(1)
                        return false; Θ(1)
                }
        }
                        return true; Θ(1)
                }
T(n) = Θ(n)


protected void incrementID(){
        systemRef.lastID++;  Θ(1)
}
T(n) = Θ(1)


protected Map<String, LinkedList<Product>> getProductsMap(){
        return systemRef.products; Θ(1)
}
T(n) = Θ(1)


protected int getID(){
        return systemRef.lastID; Θ(1)
}
T(n) = Θ(1)


protected void updateOrders(int idValue, int situation){
        systemRef.UnproccessedOrders.put(idValue, situation); Θ(1)
```

```
}
T(n) = Θ(1)


protected HashMap<Integer, Integer> getOrders(){
        return (HashMap)systemRef.UnproccessedOrders; Θ(1)
}
T(n) = Θ(1)


protected ArrayList<BinarySearchTree<Product>> getProducts() {
        return systemRef.productsOrdered; Θ(1)
}
T(n) = Θ(1)


protected void changePass (String newPass) throws InvalidClassException {
        if (getClass().equals(Seller.class)) Θ(1)
                systemRef.Sellers.putIfAbsent(username, newPass); Θ(1)
        else if (getClass().equals(Customer.class)) Θ(1)
                systemRef.Customers.putIfAbsent(username, newPass); Θ(1)
        else if (getClass().equals(Admin.class)) Θ(1)
                systemRef.Admins.putIfAbsent(username, newPass); Θ(1)
        else throw new InvalidClassException("This user is not allowed in the system"); Θ(1)
}
T(n) = Θ(1)


protected int getInputInt(Scanner scan, String loopMsg) {
        while (true) {
                System.out.print(loopMsg); Θ(1)
                try {
                        int in = scan.nextInt(); Θ(1)
                        scan.nextLine(); Θ(1)
                        return in; Θ(1)
                } catch (InputMismatchException e2) {
                        scan.nextLine(); Θ(1)
                        Sytem.out.print("\033[2A\r\033[JInvalid Input\n"); Θ(1)
                }
        }
T(n) = Θ(1)
```

ECommerceSystem Class Methods:

n = number of products in the system
s = number of sellers of the product
u = number of users in the system
r = number of requests in the system
S = number of sellers in the system
C = number of customers in the system
A = number of admins in the system

R = total number of order records

```
private void createBST() {
        productsOrdered = new ArrayList(); Θ(1)
        for (Map.Entry<String, LinkedList<Product>> entry : products.entrySet()) { Θ(n)
                productsOrdered.add(new BinarySearchTree()); Θ(1)
                LinkedList<Product> temp = entry.getValue(); Θ(n)
                for (Product product : temp) Θ(s)
productsOrdered.get(productsOrdered.size() - 1).add(product.clone());
        }
}
T(n, s) = O(n*s)

public ECommerceSystem() {
        //READING FROM FILES
        try {
                File file = new File("Products.txt"); Θ(1)
                file.createNewFile();      Θ(1)
                Scanner reader = new Scanner(file);       Θ(1)
                while (reader.hasNextLine()) {    O(n)
                        String temp = reader.nextLine(); O(s)
                        String[] lineWords = temp.trim().split("\\s+");      O(s)
                        products.put(lineWords[0], new LinkedList());     O(logn)

                        for (int i = 1; i < lineWords.length; i += 3)             O(s)
                                products.get(lineWords[0]).add(new Product(lineWords[0],
lineWords[i], Double.parseDouble(lineWords[i + 1]), Integer.parseInt(lineWords[i + 2])));  O(max(n, s))
                }
                reader.close();               Θ(1)

                file = new File("Sellers.txt");                   Θ(1)
                file.createNewFile();                   Θ(1)
                reader = new Scanner(file);                       Θ(1)
                while (reader.hasNext()) Sellers.put(reader.next(), reader.next()); O(u^2)

                file = new File("Customers.txt"); Θ(1)
                file.createNewFile();                   Θ(1)
                reader = new Scanner(file);                       Θ(1)
                while (reader.hasNext()) Customers.put(reader.next(), reader.next());       O(u^2)

                file = new File("Admins.txt");                    Θ(1)
                file.createNewFile();                   Θ(1)
                reader = new Scanner(file);                       Θ(1)
                while (reader.hasNext()) Admins.put(reader.next(), reader.next());           O(u^2)

                file = new File("Requests.txt");   Θ(1)
                file.createNewFile();                   Θ(1)
                reader = new Scanner(file);                       Θ(1)
                while (reader.hasNext()) {                        O(r)
                        int priority = Integer.parseInt(reader.next());       Θ(1)
                        if (priority == 0) Requests.add(new SellerRequest(new Seller(reader.next(),
reader.next()))); O(logr)
                        else Requests.add(new ProductRequest(reader.next()));  O(logr)
                }

                createBST();                    O(n*s)
        } catch (Exception e) {
                System.out.println("Error during opening the file.");
                e.printStackTrace();
        }
}
                                                        T(n, s) = O(n*s*max(n, s))


public void menu() {
        boolean flag = true; Θ(1)
        System.out.println("~~~~ Welcome to E-Commerce System
~~~~\n-------------------------------------"); Θ(1)
        int choice; Θ(1)
        Scanner scan = new Scanner(System.in); Θ(1)
        do {
          try {
            System.out.println("\nMENU"); Θ(1)
            System.out.println("0 - Exit\n1 - Log in\n2 - Sign up"); Θ(1)
            System.out.printf("Choice : "); Θ(1)
            choice = scan.nextInt(); Θ(1)
            scan.nextLine(); Θ(1)

            if (choice == 0) { Θ(1)
              System.out.println("\nGOOD-BYE!"); Θ(1)
              exit(); Θ(1)
              flag = false; Θ(1)
            } else if (choice == 1) { Θ(1)
              boolean logInFlag = true; Θ(1)
              do {
                try {
                  System.out.println("\n0 - Back\n1 - Log in as admin\n2 - Log in as customer\n3 - Log in
as seller"); Θ(1)
                  System.out.printf("Choice : "); Θ(1)
                  choice = scan.nextInt(); Θ(1)
                  scan.nextLine(); Θ(1)

                  if (choice == 0) logInFlag = false; Θ(1)
                  else if (choice == 1) { Θ(1)
                    System.out.printf("\nUsername: "); Θ(1)
                    String usernameValue = scan.nextLine(); Θ(1)
                    System.out.printf("Password: "); Θ(1)
                    String passwordValue = scan.nextLine(); Θ(1)
                    usernameValue.trim(); Θ(1)
                    passwordValue.trim(); Θ(1)
```

```
              if (Admins.containsKey(usernameValue) &&
Admins.get(usernameValue).equals(passwordValue)) { Θ(1)
                Admin newAdmin = new Admin(usernameValue, this); Θ(1)
                newAdmin.UI(); Θ(1)
                logInFlag = false; Θ(1)
              } else System.out.println("Invalid username or password!"); Θ(1)
            } else if (choice == 2) { Θ(1)
              System.out.printf("\nUsername: "); Θ(1)
              String usernameValue = scan.nextLine(); Θ(1)
              System.out.printf("Password: "); Θ(1)
              String passwordValue = scan.nextLine(); Θ(1)
              usernameValue.trim(); Θ(1)
              passwordValue.trim(); Θ(1)

              if (Customers.containsKey(usernameValue) &&
Customers.get(usernameValue).equals(passwordValue)) { Θ(1)
                Customer newCustomer = new Customer(usernameValue, this); Θ(1)
                newCustomer.UI(); Θ(1)
                logInFlag = false; Θ(1)
              } else System.out.println("Invalid username or password!"); Θ(1)
            } else if (choice == 3) { Θ(1)
              System.out.printf("\nUsername: "); Θ(1)
              String usernameValue = scan.nextLine(); Θ(1)
              System.out.printf("Password: "); Θ(1)
              String passwordValue = scan.nextLine(); Θ(1)
              usernameValue.trim(); Θ(1)
              passwordValue.trim(); Θ(1)

              if (Sellers.containsKey(usernameValue) &&
Sellers.get(usernameValue).equals(passwordValue)) { Θ(1)
                Seller newSeller = new Seller(usernameValue, this); Θ(1)
                newSeller.UI(); Θ(1)
                saveRequests(); O(r)
                logInFlag = false; Θ(1)
              } else System.out.println("Invalid username or password!"); Θ(1)
            } else System.out.println("Invalid choice, please try again."); Θ(1)
          } catch (Exception e) {
            scan.nextLine(); Θ(1)
            System.out.printf("Error: %s\n", e); Θ(1)
            e.printStackTrace(); Θ(1)
          }
        } while (logInFlag);
      } else if (choice == 2) { Θ(1)
        boolean signUpFlag = true; Θ(1)
        do {
          try {
            System.out.println("\n0 - Back\n1 - Sign up as customer\n2 - Sign up as seller"); Θ(1)
            System.out.printf("Choice : "); Θ(1)
            choice = scan.nextInt(); Θ(1)
            scan.nextLine(); Θ(1)

            if (choice == 0) signUpFlag = false; Θ(1)
```

```
                    else if (choice == 1) { Θ(1)
                      System.out.printf("\nUsername: "); Θ(1)
                      String usernameValue = scan.nextLine(); Θ(1)
                      System.out.printf("Password: "); Θ(1)
                      String passwordValue = scan.nextLine(); Θ(1)

                      if (!Customers.containsKey(usernameValue)) { Θ(1)
                        Customers.put(usernameValue, passwordValue); Θ(1)
                        System.out.println("You have signed up successfully.\n"); Θ(1)
                        saveCustomers(); O(C)
                        signUpFlag = false; Θ(1)
                      } else System.out.println("This username has been taken."); Θ(1)
                    } else if (choice == 2) { Θ(1)
                      System.out.printf("\nUsername: "); Θ(1)
                      String usernameValue = scan.nextLine(); Θ(1)
                      System.out.printf("Password: "); Θ(1)
                      String passwordValue = scan.nextLine(); Θ(1)
                      usernameValue.trim(); Θ(1)
                      passwordValue.trim(); Θ(1)

                      if (!Sellers.containsKey(usernameValue)) { Θ(1)
                        SellerRequest newRequest = new SellerRequest(new Seller(usernameValue, this),
passwordValue); Θ(1)
                        Requests.add(newRequest);  O(log(n))
                        System.out.println("Approval sent!\n");
                        saveRequests(); O(r)
                        signUpFlag = false; Θ(1)
                      } else System.out.println("This username has been taken."); Θ(1)
                    } else System.out.println("Invalid choice, please try again."); Θ(1)
                  } catch (Exception e) {
                    scan.nextLine(); Θ(1)
                    System.out.printf("Error: %s\n", e); Θ(1)
                    e.printStackTrace(); Θ(1)
                  }
                } while (signUpFlag);
              } else System.out.println("Invalid choice, please try again."); Θ(1)
            } catch (Exception e) {
              scan.nextLine(); Θ(1)
              System.out.printf("Error: %s\n", e); Θ(1)
              e.printStackTrace(); Θ(1)
            }
          } while (flag);
        }
T(r, s,C) = O(max(logn, s,C))


private void addProduct(String productName) {
        if (!products.containsKey(productName)) { Θ(1)
                products.put(productName, new LinkedList()); Θ(1)
                createBST(); O(n*s)
        } else System.out.println("This product already exists."); Θ(1)
        }
```

T(n, s) = O(n*s)

```
private void removeProduct(String productName) { Θ(1)
        if (products.containsKey(productName)) { Θ(1)
                products.remove(productName); O(n)
                createBST(); O(n*s)
        } else System.out.println("This product doesn't exist.");   Θ(1)
}
T(n, s) = O(n*s)
```

```
private void saveProducts() {
        try {
                FileWriter writer = new FileWriter(resourcesDir + "Products.txt"); Θ(1)

        for (Map.Entry<String, LinkedList<Product>> entry : products.entrySet()) { Θ(n)
                        writer.write(entry.getKey() + " "); Θ(1)
                        LinkedList<Product> temp = entry.getValue(); Θ(1)
                        Iterator<Product> iter = temp.iterator(); Θ(1)
                        while (iter.hasNext()) writer.write(iter.next() + " "); Θ(s)
                        writer.write("\n"); Θ(1)
                }

                writer.close(); Θ(1)
        } catch (IOException e) {
                System.out.println(e); Θ(1)
                e.printStackTrace(); Θ(1)
        }
}
T(n, s) = O(n*s)
```

```
private void saveCustomers() {
        try {
                File file = new File(resourcesDir + "Customers.txt"); Θ(1)
                file.createNewFile(); Θ(1)
                BufferedWriter bf = new BufferedWriter(new FileWriter(file)); Θ(1)

                for (Map.Entry<String, String> entry : Customers.entrySet()) { Θ(C)
                        bf.write(entry.getKey() + " " + entry.getValue()); Θ(1)
                        bf.newLine(); Θ(1)
                }

                bf.flush(); Θ(1)
                bf.close(); Θ(1)
        } catch (Exception e) {
                e.printStackTrace(); Θ(1)
        }
}

T(C) = O(C)
```

```
private void saveSellers() {
        try {
                File file = new File(resourcesDir + "Sellers.txt"); Θ(1)
                file.createNewFile(); Θ(1)
                BufferedWriter bf = new BufferedWriter(new FileWriter(file)); Θ(1)

                for (Map.Entry<String, String> entry : Sellers.entrySet()) { Θ(S)
                        bf.write(entry.getKey() + " " + entry.getValue()); Θ(1)
                        bf.newLine(); Θ(1)
                }

                bf.flush(); Θ(1)
                bf.close(); Θ(1)
        } catch (Exception e) {
                e.printStackTrace(); Θ(1)
        }
}

T(S) = O(S)


private void saveRequests() {
try {
        FileWriter writer = new FileWriter(resourcesDir + "Requests.txt"); Θ(1)
        Iterator<Request> iterator = Requests.iterator(); Θ(1)

while (iterator.hasNext()) { Θ(r)
        Request temp = iterator.next(); Θ(1)
        if (temp.priority == 0) Θ(1)
                writer.write(temp.priority + " " + ((SellerRequest) temp).user.username + " " +
((SellerRequest) temp).password + "\n"); Θ(1)
                        else writer.write(temp.priority + " " + ((ProductRequest) temp).productName
+ "\n"); Θ(1)
                }

                writer.close(); Θ(1)
        } catch (Exception e) {
                e.printStackTrace(); Θ(1)
        }
}
T(r) = O(r)


private void saveOrders(){
        try{
                FileWriter writer = new FileWriter(resourcesDir + "Orders.txt"); Θ(1)
                writer.write(lastID + "\n"); Θ(1)

        for (Map.Entry<Integer, Integer> entry : UnproccessedOrders.entrySet()){ Θ(R)
                        writer.write(entry.getKey() + " " + entry.getValue() + "\n");  Θ(1)
                }

                writer.close(); Θ(1)
        }
        catch (Exception e){
                e.printStackTrace(); Θ(1)
        }
}
T(R) = O(R)



private void exit() {
                saveProducts(); O(n*s)
                saveCustomers(); O(C)
                saveSellers(); O(S)
                saveRequests(); O(r)
                saveOrders(); O(R)
        }

T(n,s,C,S,r,R) = O(max(n*s,C,S,r,R))
```

Product Inner Static Class Methods:

```
public Product(String productName, String sellerName, double price, int stock) {
        this.productName = productName; Θ(1)
        this.sellerName = sellerName; Θ(1)
        this.price = price; Θ(1)
        this.stock = stock;  Θ(1)
}
T(n) =Θ(1)


public String toString(){
        StringBuilder strb = new StringBuilder(); Θ(1)
        strb.append(sellerName).append(" ")
                        .append(price).append(" ")
                        .append(stock); Θ(n)

                        return strb.toString(); Θ(1)
                }
T(n) =Θ(n)


public int compareTo(Product o){
        return Double.compare(price, o.price); Θ(1)
}
T(n) =Θ(1)

public String getProductName() {
        return productName; Θ(1)
}
T(n) =Θ(1)

public int getStock() {
        return stock; Θ(1)
}
T(n) =Θ(1)


public void setStock(int stock) {
        this.stock = stock; Θ(1)
}
T(n) =Θ(1)

public double getPrice() {
        return price; Θ(1)
}
T(n) =Θ(1)
public Product clone() {
        try {
                Product copy = (Product) super.clone(); Θ(1)
                //gives me the Product object which is shallow copied
                return copy; Θ(1)
        } catch (CloneNotSupportedException e) {
                //this will never happen
                return null; Θ(1)
        }
}

T(n) =Θ(1)
```

Admin Inner Class Methods:

```
public Admin(String usernameValue, ECommerceSystem callerSystem) {
        super(usernameValue, callerSystem); Θ(1)
}
T(n) =Θ(1)


public void UI() {
        int choice; Θ(1)
        boolean flag = true; Θ(1)
        Scanner scan = new Scanner(System.in); Θ(1)

        System.out.printf("\nWelcome dear %s!", username); Θ(1)

        do {
                try {
                        System.out.println("\nWhat do you want to do?"); Θ(1)
```

```
            System.out.println("0 - Exit\n1 - Check requests\n2 - Remove user\n3 - Add
product\n4 - Remove product"); Θ(1)
            System.out.printf("Choice: "); Θ(1)
            choice = scan.nextInt(); Θ(1)
            scan.nextLine(); Θ(1)

            if (choice == 0) { Θ(1)
                System.out.printf("Goodbye %s!\n", username); Θ(1)
                flag = false; Θ(1)
            } else if (choice == 1) { Θ(1)
                if (Requests.size() == 0) System.out.println("There is no pending
request."); Θ(1)
                else {
                    boolean acceptFlag = true; Θ(1)
                    do {
                        System.out.printf("Next pending request --> %s\n",
Requests.peek()); Θ(1)
                        System.out.printf("Do you want to accept it? (y: yes,
n: no) --> "); Θ(1)
                        String acceptance = scan.nextLine(); Θ(1)
                        if (acceptance.equals("y")) { Θ(1)
                            if (Requests.peek().priority == 0) Θ(1)
                                Sellers.put(((SellerRequest)
Requests.peek()).user.username, ((SellerRequest) Requests.poll()).password); Θ(logr)
                            else addProduct(((ProductRequest)
Requests.poll()).productName);  Θ(logr)* O(n*s)
                            acceptFlag = false; Θ(1)
                        } else if (acceptance.equals("n")) { Θ(1)
                            Requests.poll(); Θ(logr)
                            acceptFlag = false; Θ(1)
                        } else System.out.println("Invalid choice, please try
again."); Θ(1)
                    } while (acceptFlag);
                }
            } else if (choice == 2) { Θ(1)
                System.out.printf("Enter the type of the user you want to remove (s:
seller, c: customer): "); Θ(1)
                String remove = scan.nextLine(); Θ(1)
                if (remove.equals("s")) { Θ(1)
                    System.out.printf("Enter the username you want to remove:
"); Θ(1)
                    remove = scan.nextLine(); Θ(1)
                    remove.trim(); Θ(1)
                    if (Sellers.containsKey(remove)){ Θ(1)
                        Sellers.remove(remove); Θ(1)
                        File file = new File(resourcesDir + "Sellers/" + remove
+ ".txt"); Θ(1)
                        file.delete(); Θ(1)
                    }
                    else System.out.println("There is no such user."); Θ(1)
                } else if (remove.equals("c")) { Θ(1)
```

```
                                                System.out.printf("Enter the username you want to remove: 
"); Θ(1)
                                                remove = scan.nextLine(); Θ(1)
                                                remove.trim(); Θ(1)
                                                if (Customers.containsKey(remove)){ Θ(1)
                                                        Customers.remove(remove); Θ(1)
                                                        File file = new File(resourcesDir + "Customers/" + 
remove + ".txt"); Θ(1)
                                                        file.delete(); Θ(1)
                                                }
                                                else System.out.println("There is no such user."); Θ(1)
                                        } else System.out.println("Invalid choice."); Θ(1)
                                } else if (choice == 3) { Θ(1)
                                        System.out.printf("Enter the name of the product: "); Θ(1)
                                        String productName = scan.nextLine(); Θ(1)
                                        productName.trim(); Θ(1)
                                        addProduct(productName); O(n*s)
                                } else if (choice == 4) { Θ(1)
                                        System.out.printf("Enter the name of the product: "); Θ(1)
                                        String productName = scan.nextLine(); Θ(1)
                                        productName.trim(); Θ(1)
                                        removeProduct(productName); O(n*s)
                                } else System.out.println("Invalid choice, please try again."); Θ(1)
                        } catch (InputMismatchException e) {
                                scan.nextLine(); Θ(1)
                                System.out.printf("Error: %s\n", e); Θ(1)
                                e.printStackTrace(); Θ(1)
                        } catch (Exception e) {
                                System.out.printf("Error: %s\n", e); Θ(1)
                                e.printStackTrace(); Θ(1)
                        }
                } while (flag);
}
}
```

T(n,s,r) = O(logr*n*s)

Request Inner Abstract Static Class Methods:

```
public int compareTo(Request o) {
        return priority - o.priority; Θ(1)
}
```

T(n)=Θ(1)

SellerRequest Inner Static Class Methods:

```
public SellerRequest(User user, String password) {
        this.user = user; Θ(1)
        this.password = password; Θ(1)
        priority = 0; Θ(1)
}
```

T(n)=Θ(1)

```
public String toString() {
        return "Seller request: " + user.username; Θ(1)
}
```

T(n)=Θ(1)

ProductRequest Inner Static Class Methods:

```
public ProductRequest(String name) {
                productName = name; Θ(1)
                priority = 1; Θ(1)
        }
```

T(n)=Θ(1)

```
public String toString() {
        return "Product request: " + productName; Θ(1)
}
```

T(n)=Θ(1)