**Add client**

1. Register member feature, because you’re adding an object which represents a person, one at a time
2. Steps to perform
   1. Ask for data from user/client
   2. Check if client already exists
   3. Create new client object
   4. Add object to client list
   5. Return result of operation
3. Design choices
   1. Where should client uniqueness be validated? Uniqueness should either be validated by the system or by the client list. We chose the client list because it is simpler to have the client list search through its own attributes rather than having to have the system get the list of clients from the client list then search through it.
   2. Where should the ID for clients be generated? The generation of an ID can occur in the system or in the client object. We chose to have the client object generate the ID whenever a new object is instantiated. We chose to have the client generate the ID so that the system doesn’t become too tightly coupled with the client class.
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information from the user.
   2. The database system is responsible for propagating the user request to the back-end system
   3. ClientList is the class that will hold all the clients as well as validate that a client is unique
   4. Client is the class that will hold a single client's information namely, clientID, name and address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user | UI | DB system | Client list | Client 1 |
| 1. Request op |  |  |  |  |
|  | 2. ask for data |  |  |  |
| 3. input data |  |  |  |  |
|  |  | 5. searchList(name, address) |  |  |
|  |  |  | 6. Return ref or null |  |
|  | 7. if client exists return error, else continue |  |  |  |
|  |  | 8. createClient(name, address) |  |  |
|  |  |  |  | 9. genID() |
|  |  |  |  | 10. return true/false |
|  |  | 11. Insertclient(client1) |  |  |
|  |  |  | 12.Return true/false |  |
|  |  | 14. Return client1 |  |  |
|  | 15. Return Client 1 information |  |  |  |

**Add products**

1. Addbooks, because we’re changing the inventory (catalogue) of the warehouse(library) by adding new objects
2. Steps to perform
   1. Ask for product info
   2. Check if product exists
   3. Create new product
   4. Add product to inventory
   5. Return product info
   6. Ask if loop, repeat as necessary
3. Design choices
   1. Where and by what should the looping be handled? The looping starts after the user puts in the request to add a product, this occurs in the user interface. Once the product is added the loop begins again by asking the user if they want to add more products.
   2. What verifies the uniqueness of products? This could be done by the UI, the System, or the Inventory. Since the database system is making decisions, it should handle duplicates.
   3. How should the system respond when a duplicate is found? The options are either, return an error for the product, overwrite the product, allow duplicates, or update the quantity. Instead of making a new product the quantity of the provided product is added onto the existing products quantity in the list. Since we need user input, the UI handles this case in tandem with the system.
   4. How are price changes handled? This could be done by the UI or the System. Whenever a product is input that matches an existing product in the inventory list, but the sales price is different. The user should be prompted if they want to change the sales price for all those products in the inventory list. So, this operation should be handled by the UI since it interfaces directly with the user.
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information from the user.
   2. The database system is responsible for propagating the user request to the back-end system
   3. The inventory is responsible for maintaining a list of all the products that the warehouse has/ previously had. The inventory is also responsible for verifying if an input product matches one already in the inventory.
   4. product is the class that will hold a single product information namely, product name, price, quantity, and productId,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | UI | DB System | Inventory | Product1 |
| 1. request op |  |  |  |  |
|  | 2. ask for data |  |  |  |
| 3. input data |  |  |  |  |
|  |  | 4. SearchInventory(name, description) |  |  |
|  |  |  | 5. Return ref or null |  |
|  | 6. If ref returned, ask if price or quantity change, else go to 13 |  |  |  |
| 7. input response and new price or new quantity or both |  |  |  |  |
|  |  | 8. if new price, editPrice(pid, newPrice), |  |  |
|  |  |  |  | 9. Return true/false |
|  |  | 10. If new quantity, editQuantity(pid, newQuant) |  |  |
|  |  |  |  | 11. Return true/false |
|  | 12. display change and go to 19 |  |  |  |
|  |  | 13. CreateProduct(name, desc, price, quant) |  |  |
|  |  |  |  | 14. Return true/false |
|  |  | 15. InsertItem(product1) |  |  |
|  |  |  | 16. Return true/false |  |
|  |  | 17. Return product1 |  |  |
|  | 18. Display prod1 |  |  |  |
|  | 19. Ask if more items to add |  |  |  |
| 20. Input response |  |  |  |  |
|  | 21. If yes go to 2, if no, exit to menu |  |  |  |

**Add product to a Wishlist & Change Wishlist quantity if product is already in list**

1. Issue books, because we need to search for an available product(book) and if it exists add it to the member’s booklist (Wishlist).
2. Steps to perform
   1. Request the product that the user wants and their clientID.
   2. Check to see if that product exists.
   3. If it does exist add it to the Wishlist.
3. Design choices
   1. What should be done if they try to add a product that’s already on the list? Could be handled by the system, the UI or the wishlist itself. If there is already a wish for the same product, the UI should prompt the user to change the quantity of the wish, rather than creating a duplicate wish.
   2. Should clients be allowed to add products with zero stock? We are going to allow the client to add a product with zero stock into their Wishlist because they haven’t ordered it yet and if they do it will go onto the waitlist. This therefore doesn’t need to be checked prior to adding the product.
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information from the user.
   2. The database system is responsible for propagating the user request to the back-end system
   3. The inventory is responsible for maintaining a list of all the products that the warehouse has/ previously had. The inventory is also responsible for verifying if an input product matches one already in the inventory.
   4. ClientList is the client storage that is used to get the client that the user specified to add an item to their Wishlist.
   5. Wishlist is a client unique list of wishes that the client may or not purchase in the future.
   6. Wish is a single product and quantity that is stored in the clients Wishlist.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| User | UI | DB System | Inventory | Client list | Client1.wishlist | wish1 |
| 1. request op |  |  |  |  |  |  |
|  | 2. ask for data |  |  |  |  |  |
| 3. input data |  |  |  |  |  |  |
|  |  | 4. SearchInventory(pid) |  |  |  |  |
|  |  |  | 5. return t/f |  |  |  |
|  | 6. if false, go to 2, else continue |  |  |  |  |  |
|  |  | 7. searchClientList(cid) |  |  |  |  |
|  |  |  |  | 8. return client1 or null |  |  |
|  |  | 9. searchWishlist(pid) |  |  |  |  |
|  |  |  |  |  | 10. Return wish1 or null |  |
|  | 11. If true, ask if change quantity, else go to 16 |  |  |  |  |  |
| 12. Enter response |  |  |  |  |  |  |
|  |  | 13. If yes, EditWish(newQuant), else go to 22 |  |  |  |  |
|  |  |  |  |  |  | 14. Return t/f |
|  | 15. Display wish1, go to 22 |  |  |  |  |  |
|  |  | 16. CreateWish(pid, quant) |  |  |  |  |
|  |  |  |  |  |  | 17. Return t/f |
|  |  | 18. InsertWish(wish1) |  |  |  |  |
|  |  |  |  |  | 19. Return t/f |  |
|  |  | 20. Return wish1 |  |  |  |  |
|  | 21. Display wish1 |  |  |  |  |  |
|  | 22. Ask if add another product |  |  |  |  |  |
| 23. Enter response |  |  |  |  |  |  |
|  | 24. If yes, go to 2, else end |  |  |  |  |  |

**Display all clients with details**

1. Get Transactions, because those use cases request all the information about a particular client. This use case will display all information about a client.
2. Steps to perform
   1. The user will issue a request for all the client information.
   2. The system will process this request.
   3. The system will look through the entire clientlist
   4. At each client it displays the client's information back to the user.
3. Design choices
   1. What information should be displayed? We decided that only the client's personal information is going to be displayed namely, clientID, name, address and whether their Wishlist is empty.
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information/requests from the user.
   2. The database system is responsible for propagating the user request to the back-end system
   3. The clientlist is the collection class of all the clients, it doesn’t hold all the client’s information, but it holds all the client objects which hold the client’s information.
   4. The client class holds the information of a single client, this is where all the information that user requests is going to come from.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | UI | DB System | ClientList | Client1 |
| 1. request all client info op |  |  |  |  |
|  |  | 2.GetAllClients() |  |  |
|  |  |  | 3. Return a list of all clients |  |
|  |  | 4. Begin Loop for each client |  |  |
|  |  | 5. getClientInfo(Client1) |  |  |
|  |  |  |  | 6. return info |
|  | 7. display client info |  |  |  |
|  |  | 8. End Loop once all clients are displayed |  |  |

**Display all products with details**

1. Get transactions, for the same reason as above, display all info in a category/list
2. Steps to Perform
   1. User issues request for inventory listing
   2. System processes
   3. System scans the inventory
   4. System prints product id, name, description, quantity avail, price
3. Design Choices
   1. What information should be displayed? We decided that the information that should be displayed about a product is the product id, name, description, quantity, and price.
   2. What class should handle the information retrieval? Could be done by the inventory or the system. We chose for the system to get information, as the inventory serves as a list class and therefore should not be concerned with the details of a product to increase its cohesion
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information from the user.
   2. The database system is responsible for propagating the user request to the back-end system
   3. The inventory is responsible for maintaining a list of all the products that the warehouse has/ previously had. The system will get the list of products from the inventory
   4. The product is a single item that is for sale, this class holds the information that the user wants displayed to them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | UI | DB System | Inventory | Product1 |
| 1. request products details |  |  |  |  |
|  |  | 2. printCatalogue() |  |  |
|  |  |  | 3. returns an item list |  |
|  |  | 4. loop for each product |  |  |
|  |  | 5. getProductInfo(prod1) |  |  |
|  |  |  |  | 6. return info |
|  | 7. display product info |  |  |  |
|  |  | 8. end loop once all products displayed |  |  |

**Display all products in a specified client’s Wishlist**

1. Get transactions, because the user must specify a client(member) and the system will get the Wishlist (transaction list) from that client.
2. Steps to Perform
   1. The user will issue a request for a client’s Wishlist
   2. The user is prompted for a clientID
   3. The system will search through the clientlist for that client
   4. The system will then get that clients Wishlist
   5. Then for each wish in that list, it is displayed to the user
3. Design Choices
   1. Should Wishlist entries be product references? This could either be handled by the product class or a new class called wish. We determined that it is necessary to have a wish be its own class and to not just borrow the product class. We did this because the product is used on the warehouse side, and we don’t want to mix the client side and warehouse side as this would lower the cohesion of the product class.
4. Distribution of Process Among Software Classes
   1. The user interface is responsible for retrieval of information from the user.
   2. The database system is responsible for propagating the user request to the back-end system and information from the back end to the user.
   3. The clientlist is the collection class of all the clients, it doesn’t hold all the client’s information, but it holds all the client objects which hold the client’s information. The system must look through this list for their intended client.
   4. The Wishlist contains all the wishes that a client may or may not buy in the future. System will have to get this Wishlist to display the contents of it.
   5. The wish class contains information about a product that the client may buy. This holds all the information that the user is requesting.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| User | UI | DB System | Clientlist | Client1.wishlist | wish1 |
| 1. request op |  |  |  |  |  |
|  | 2.Prompt for client ID |  |  |  |  |
| 3.Input client id |  |  |  |  |  |
|  |  | 4.SearchClientList(cid) |  |  |  |
|  |  |  | 5.Return client1 or null |  |  |
|  | 6.If null, print error, else continue |  |  |  |  |
|  |  | 7.Get wishlist() |  |  |  |
|  |  |  |  | 8.Return client1.wishlist or null |  |
|  | 9.If null print empty wishlist msg, else continue |  |  |  |  |
|  |  | 10. Loop for each wish |  |  |  |
|  |  | 11. Getwishinfo(wish1) |  |  |  |
|  |  |  |  |  | 12.Returns info |
|  | 13. Display info |  |  |  |  |
|  |  | 14. End loop after each wish is displayed |  |  |  |