Method Specifications

Computer.cs:

```
/*
         * @param: size, which would be 1,2 or 3 based on the admins input
         * @return: void
         * initializes the catalogue and inventory based on the size
         * */
        private void initializeInventory(int size)

/*
         * @return: void
         * initialize robots for our warehouse and add them to the robots list
         * */
        private void initializeRobots()

/*
         * @return: void
         * initialize delivery and restocking trucks for our warehouse
         * */
        private void initializeTrucks()

/*
         * @return: void
         * initialize shelves for our warehouse based on warehouse layout. Generate
unique id's for each shelf
         * */
        private void initializeShelves()

/*
         * @return: void
         * goes through the inventory and adds the items to the shelves items list, and
updates invID with the number of items we started with
         * */
        public void loadShelves()

/*
         * @param: an Order to be fulfilled by the robots
         * @return: void
         * fulfills a client's order by first adding it to the orders log and then
calling collect order to get the items
         * */
        public void fulfillOrder(Order order)

/*
         * @param: order to be collected by the robots
         * @return: void
         * collects a client's order by converting the order to a list of items and calls
robots on new threads to collect those items
         * after collection, add the order to the processed orders queue once the items
have been collected and inventory is updated
         * */
        private void collectOrder(Order order)

/*
         * @param: an order to be validated
```

```
         * @return: boolean
         * validates an order by ensuring all items are in inventory and stock is
available
         * */
        public bool OrderIsValid(Order order)

/*
         * @param: an Order
         * @return: a List of Items
         * Goes through the order and creates a list of individual items that are within
that order
         * */
        private List<Item> orderToItems(Order order)

/*
         * @param: a delivery truck
         * @return: void
         * loads processed orders onto a truck and sends truck out for delivery
         * */
        public void loadProcessedOrders(DeliveryTruck currTruck)

/*
         * @param: a bool indicating if a new truck is needed
         * @return: Delivery truck
         * services the restocking trucks in the docking queue and adds a delivery truck
to the docking queue if asked for
         * */
        public DeliveryTruck serviceNextTruck(bool needNewDeliveryTruck = false)

/*
         * @param: delivery truck
         * @return: void
         * sends the delivery truck out to deliver the orders to the clients
         * */
        public void deliverOrders(DeliveryTruck truck)

/*
         * @param: restock truck
         * @return: void
         * goes through the restock truck and calls robots to restock the items and
updates the catalogue
         * */
        public void RestockTruckItems(RestockTruck truck)

/*
         * @param: an item to be restocked in inventory
         * @return: void
         * restocks item to a shelf. Shelf id generated using "Random" class. If shelf
full, try new shelf. Loop until empty shelf found
         * */
        public void restockItem(Item newItem)

/*
         * @return: string with ids of the restock trucks bringing the items
         * reads inventory to figure out which items are below their max stock and calls
restock truck to bring items to warehouse
         * */
        public string ReadAndReplaceCatalogStock()
```

```
/*
         * @param: product to add to catalogue
         * @return: void
         * adds the new product to the catalogue if the product with that name is not
already in it.
         * */
        public static void AddNewCatalogItem(Product item)

/*
         * @param: Order for which the notification is required
         * @return: string
         * returns the status of the order for which it was requested
         * */
        public string notifyUser(Order order)

/*
         * @param: Order for which the status is getting updated
         * @param: status
         * @return: void
         * updates the status for the order given in the order log
         * */
        public void setOrderStatus(Order order, String status)

/*
         * @param: Order for which the notification is required
         * @return: string
         * returns the status of the order for which it was requested
         * */
        public String queryOrderStatus(Order order)

/*
         * @param: List of items to update the inventory with
         * @return: void
         * updates the inventory file using json serialize
         * */
        public static void UpdateInventory(List<Item> newItems)

/*
         * @param:
         * @return: List of items from the inventory
         * reads the inventory and returns the items from the inventory using json
deserialize
         * */
        public static List<Item> ReadInventory()

/*
         * @param: Array of products to update the catalogue
         * @return: void
         * updates the catalogue file using json serialize
         * */
        public static void UpdateCatalog(Product[] newItems)

/*
         * @param:
         * @return: array of products from the catalogue
         * reads the catalogue and returns the products from the catalogue using json
deserialize
         * */
        public static Product[] ReadCatalog()
```

```
/*
         * @param: product the admin wants to discontinue
         * @return: void
         * removes all of the stock of that product from the warehouse using collect
order
         * */
        public void discontiueProduct(Product product)
```

Admin.cs

```
/*
         * @return: void
         * Continually run the admin console
         */
        public void startAdmin()

/*
         * @return: void
         * Displays console options and allows the admin to select from a menu
         */
        public void displayAdmin()
/*
         * @return: void
         * Display all past and present orders along with their status
         */
        public void viewOrders()

/*
         * @return: void
         * view all stock of products in JSON database
         */
        public void viewStock()

/*
         * @param: order
         * @return: void
         * send the client's order to the warehouse to fullfil, and load trucks for the
processed orders
         */
        public void sendOrder(Order newOrder)

/*
         * @param: Intervel for when the trucks should be sent out
         * @return: void
         * if the truck isn't full and the time between the orders is greater than the
interval, send out the truck
         */
        public static void deliveryTimer(int interval)

/*
         * @return: void
         * Ouptuts an alert and then calls on the warehouse to replace all items that are
below max capacity
         */
        public void notifyAdmin()
```

Server.cs

```
/*
         * @return: void
         * Creates a Server socket and begins listening for client connections
         * */
        private static void SetupServer()

/*
         * @return: void
         * Close all connected client (we do not need to shutdown the server socket as
its connections
         * are already closed with the clients).
         */
        private static void CloseAllSockets()

/*
         * @return: void
         * Accepts a client and if no errors, will begin to get data
         */
        private static void AcceptCallback(IAsyncResult AR)

/*
         * @return: void
         * Recieves and processes client data. Can either send back JSON data or forward
an order to a warehouse
         */
        private static void ReceiveCallback(IAsyncResult AR)
```

Client/Program.cs

```
/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Displays store menu options
         */
        public static void displayStore(List<int> cart)

/*
         * @return: Product Object Array
         * Requests JSON data from the server to serialize for the product list
         */
        public static Product[] ReadInventory()

/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Displays store inventory and menu options
         */
        public static void viewProducts(List<int> cart)

/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Displays client's active cart and menu options
         */
        public static void viewCart(List<int> cart)
```

```
/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Adds an item the client chooses to their cart if it is not full
         */
        public static void addCart(List<int> cart)

/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Removes an item from the clients cart if cart has item to remove
         */
        public static void removeCart(List<int> cart)

/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Displays Checkout options
         */
        public static void checkout(List<int> cart)

/*
         * @return: int
         * @Param: String[] args
         * Main entry point of program
         */
        public static int Main(String[] args)

/*
         * @return: void
         * @Param: takes a clients active cart in the form of a list
         * Sends the cart to the server through sockets
         */
        public static void sendServer(List<int> cart, string cmd)

/*
         * @return: void
         * Starts up the client interface
         */
        public static void startClient()
```