Candy Factory

Practical applications of object-oriented design patterns

Authors

Travis Dagostino, Kelly Payne, Abdulraheem Giwa

Contents

SECTION 1 - Final State of System

The final state of the system represents a fully functional integration of user interface and backend components, providing a consolidated architecture that distinctly serves Customer and Administrative roles. Key implemented features include a comprehensive Customer Dashboard with a candy catalog, cart, and checkout workflows, alongside an Administration Dashboard that supports order tables, basic user management, and inventory management. To ensure robustness, data persistence and inventory management, including low-stock alerts, were operationalized using a CSV-based system. Significant changes since Projects 5 and 6 include the introduction of a Backend Facade to streamline communication between layers and the specific focus customer versus admin versions of the application. However, certain features originally planned during the earlier project phases, such as full user management lifecycles (currently only offering the ability to review users and adjust their roles), detailed analytics, and extended inventory controls, were not fully implemented in this iteration to prioritize the stability of the core transactional features.

Important note when running the UI of the application:

Admin User Credentials:

Username: admin

Password: admin123

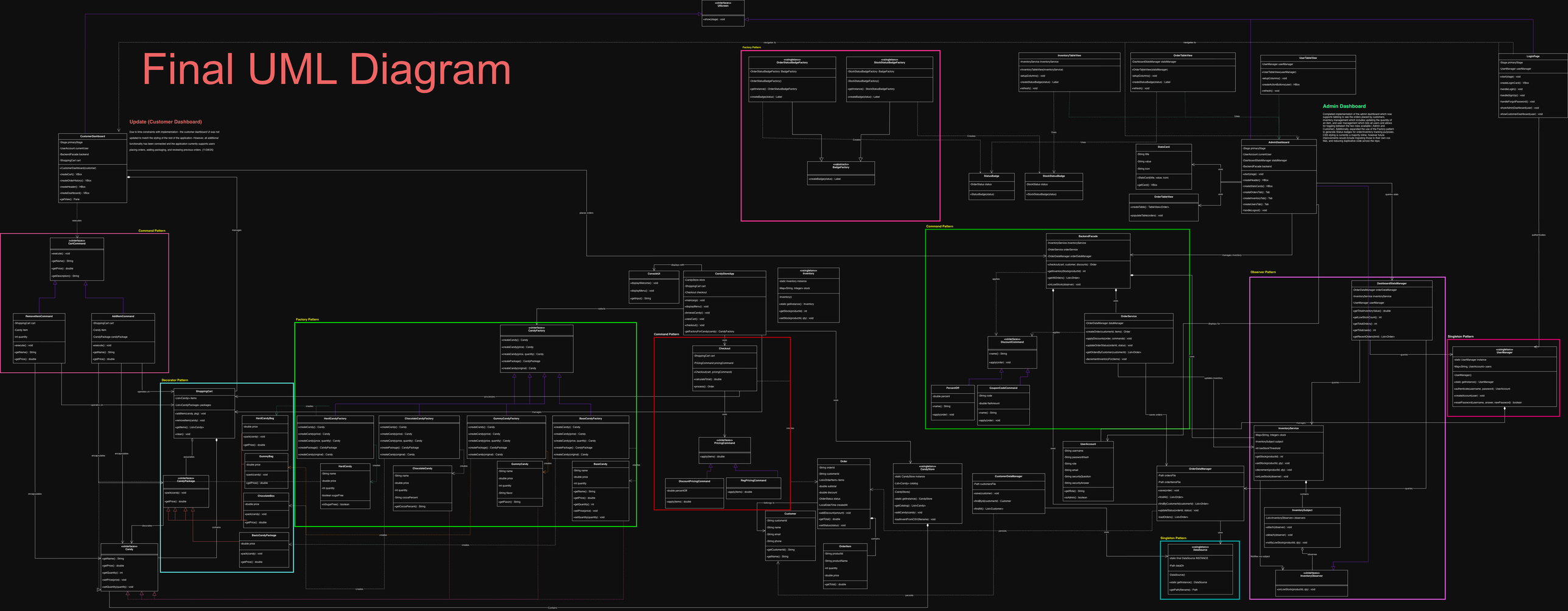
Customer Credentials:

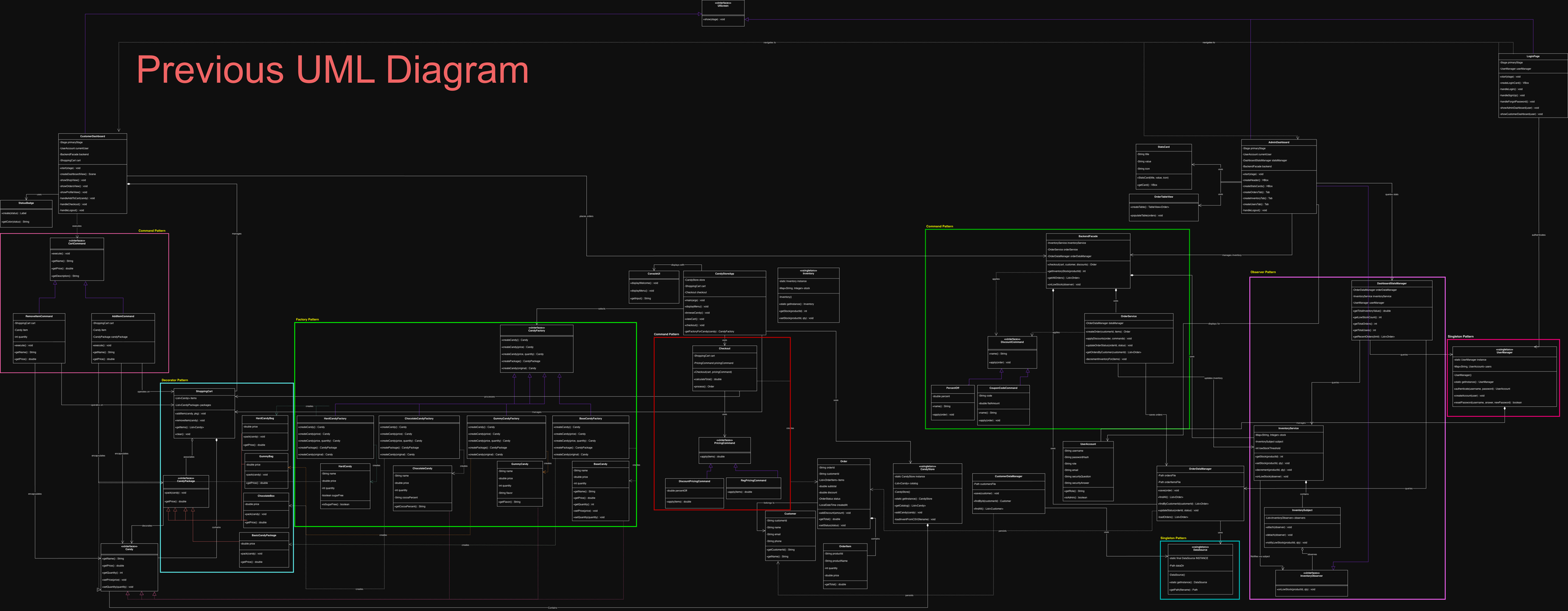
Username: customer

Password: customer123

SECTION 2 - UML Class Diagrams

The system architecture was adjusted to focus on customer and administrative roles versus creating an application focused on in-person versus virtual stores. A Backend Facade was introduced to improve the interaction between the user interface and the back-end components, resulting in a more modular structure. Inventory management was implemented using a CSV-based system, supported by fully integrated dashboards, order processing, and checkout workflows. Additionally, system reliability was strengthened through enhanced error handling, build scripts, and user management. Throughout development, additional design patterns were applied to ensure the code remains clean and maintainable including the singleton for user management/data control, factory method for UI badges for inventory/order tracking, and the command pattern for the shopping cart implementation.


Final UML Diagram
Update (Customer Dashboard)
Due to time constraints with implementation - the customer dashboard UI was not updated to match the styling of the rest of the application. However, all additional functionality has been connected and the application currently supports users placing orders, adding packaging, and reviewing previous orders. (11/28/25)
Admin Dashboard
Completed implementation of the admin dashboard which now supports tabbing to see the orders placed by customers, inventory management which includes updating the quantity of stock, and a user management system which allows admins to switch by hugging between the two roles available (Admin and Customer). Additionally, expanded the use of the Factory pattern to include styling for the different badges and stat boxes. CSS styling is currently a majority inline, however future improvements would include migrating those to their own css files, and reducing duplicative code across the repo.
«interface» UIScreen
+show(stage) : void
navigates to
Factory Pattern
«singleton» OrderStatusBadgeFactory
-OrderStatusBadgeFactory: BadgeFactory
-OrderStatusBadgeFactory()
+getInstance() : OrderStatusBadgeFactory
+createBadge(status) : Label
«singleton» StockStatusBadgeFactory
-StockStatusBadgeFactory: BadgeFactory
-StockStatusBadgeFactory()
+getInstance() : StockStatusBadgeFactory
+createBadge(status) : Label
«abstract» BadgeFactory
+createBadge(status) : Label
Creates
Creates
InventoryTableView
-inventoryService inventoryService
+InventoryTableView(inventoryService)
-setupColumns() : void
-createStatusBadge(status) : Label
+refresh() : void
OrderTableView
-DashboardStatsManager statsManager
+OrderTableView(statsManager)
-setupColumns() : void
-createStatusBadge(status) : Label
+refresh() : void
navigates to
Uses
UserTableView
-UserManager userManager
+UserTableView(userManager)
-setupColumns() : void
-createActionButtons(user) : HBox
+refresh() : void
LoginPage
-Stage primaryStage
-UserManager userManager
+start(stage) : void
-create LoginCard() : VBox
-handleLogin() : void
-handleSignUp() : void
-handleForgotPassword() : void
-showAdminDashboard(user) : void
-showCustomerDashboard(user) : void
Uses
StatusBadge
-OrderStatus status
+StatusBadge(status)
StockStatusBadge
-StockStatus status
+StockStatusBadge(status)
uses
StatsCard
-String title
-String value
-String icon
+StatsCard(title, value, icon)
+getCard() : VBox
OrderTableView
+createTable() : TableView<Order>
+populateTable(orders) : void
AdminDashboard
-Stage primaryStage
-UserAccount currentUser
-DashboardStatsManager statsManager
-BackendFacade backend
+start(stage) : void
-createHeader() : HBox
-createStatsCards() : HBox
-createOrdersTab() : Tab
-createInventoryTab() : Tab
-createUsersTab() : Tab
-handleLogout() : void
queries stats
authenticates
CustomerDashboard
-Stage primaryStage
-UserAccount currentUser
-BackendFacade backend
-ShoppingCart cart
+CustomerDashboard(customer)
-createCart() : VBox
-createOrderHistory() : VBox
-createHeader() : HBox
-createDashboard() : VBox
-getView() : Pane
executes
Command Pattern
«interface» CartCommand
+execute() : void
+getName() : String
+getPrice() : double
+getDescription() : String
RemoveItemCommand
-ShoppingCart cart
-Candy item
-int quantity
+execute() : void
+getName() : String
+getPrice() : double
AddItemCommand
-ShoppingCart cart
-Candy item
-CandyPackage candyPackage
+execute() : void
+getName() : String
+getPrice() : double
manages
places orders
displays with
ConsoleUI
+displayWelcome() : void
+displayMenu() : void
+getInput() : String
CandyStoreApp
-CandyStore store
-ShoppingCart cart
-Checkout checkout
+main(args) : void
+displayMenu() : void
+browseCandy() : void
+viewCart() : void
+checkout() : void
-getFactoryForCandy(candy) : CandyFactory
«singleton» Inventory
-static Inventory instance
-Map<String, Integer> stock
-Inventory()
+static getInstance() : Inventory
+getStock(productId) : int
+setStock(productId, qty) : void
reads/s
Factory Pattern
«abstract» CandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
HardCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
ChocolateCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
GummyCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
BaseCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
creates
Command Pattern
Checkout
-ShoppingCart cart
-PricingCommand pricingCommand
+Checkout(cart, pricingCommand)
+calculateTotal() : double
+process() : Order
uses
«interface» PricingCommand
+apply(items) : double
DiscountPricingCommand
-double percentOff
+apply(items) : double
RegPricingCommand
+apply(items) : double
processes
Order
-String orderId
-String customerId
-List<OrderItem> items
-double subtotal
-double discount
-OrderStatus status
-LocalDateTime createdAt
+addDiscount(amount) : void
+getTotal() : double
+setStatus(status) : void
belongs to
OrderItem
-String productId
-String productName
-int quantity
-double price
+getTotal() : double
contains
«singleton» CandyStore
-static CandyStore instance
-List<Candy> catalog
-CandyStore()
+static getInstance() : CandyStore
+getCatalog() : List<Candy>
+addCandy(candy) : void
-loadInventoryFromCSV(filename) : void
Customer
-String customerId
-String name
-String email
-String phone
+getCustomerId() : String
+getName() : String
persists
CustomerDataManager
-Path customersFile
+save(customer) : void
+findById(customerId) : Customer
+findAll() : List<Customer>
Decorator Pattern
«interface» CandyPackage
+getItems() : List<Candy>
+clear() : void
ShoppingCart
-List<Candy> items
-List<CandyPackage> packages
+addItem(candy, pkg) : void
+removeItem(candy) : void
+getItems() : List<Candy>
+clear() : void
operates on
encapsulates
encapsulates
encapsulates
encapsulates
decorates
contains
HardCandyBag
-double price
+pack(candy) : void
+getPrice() : double
GummyBag
-double price
+pack(candy) : void
+getPrice() : double
ChocolateBox
-double price
+pack(candy) : void
+getPrice() : double
BasicCandyPackage
-double price
+pack(candy) : void
+getPrice() : double
HardCandy
-String name
-double price
-int quantity
-boolean sugarFree
+isSugarFree() : boolean
ChocolateCandy
-String name
-double price
-int quantity
-String cocoaPercent
+getCocoaPercent() : String
GummyCandy
-String name
-double price
-int quantity
-String flavor
+getFlavor() : String
BaseCandy
-String name
-double price
-int quantity
+getName() : String
+getPrice() : double
+getQuantity() : int
+setPrice(price) : void
+setQuantity(quantity) : void
«interface» Candy
+getName() : String
+getPrice() : double
+getQuantity() : int
+setPrice(price) : void
+setQuantity(quantity) : void
Contains
BackendFacade
-InventoryService inventoryService
-OrderService orderService
-OrderDataManager orderDataManager
+checkout(cart, customer, discounts) : Order
+getInventory(Stock)productId) : int
+getAllOrders() : List<Order>
+onLowStock(observer) : void
manages inventory
applies
«interface» DiscountCommand
+name() : String
+apply(order) : void
PercentOff
-double percent
+name() : String
+apply(order) : void
CouponCodeCommand
-String code
-double flatAmount
+name() : String
+apply(order) : void
uses
OrderService
-OrderDataManager dataManager
+createOrder(customerId, items) : Order
+applyDiscounts(order, commands) : void
+updateOrderStatus(orderId, status) : void
+getOrdersByCustomer(customerId) : List<Order>
-decrementInventory For(items) : void
displays for
updates inventory
uses
saves orders
UserAccount
-String username
-String passwordHash
-String role
-String email
-String securityQuestion
-String securityAnswer
+getRole() : String
+isAdmin() : boolean
uses
OrderDataManager
-Path ordersFile
-Path orderItemsFile
+save(order) : void
+findAll() : List<Order>
+findByCustomerId(customerId) : List<Order>
+updateStatus(orderId, status) : void
-loadOrders() : List<Order>
persists
Observer Pattern
DashboardStatsManager
-OrderDataManager orderDataManager
-InventoryService inventoryService
-UserManager userManager
+getTotalInventoryValue() : double
+getLowStockCount() : int
+getTotalOrders() : int
+getTotalCanes() : int
+getRecentOrders(limit) : List<Order>
queries
InventoryService
-Map<String, Integer> stock
-InventorySubject subject
-int lowStockThreshold
+getStock(productId) : int
+setStock(productId, qty) : void
+decrement(productId, qty) : void
+onLowStock(observer) : void
notifies via subject
InventorySubject
-List<InventoryObserver> observers
+attach(observer) : void
+detach(observer) : void
+notifyLowStock(productId, qty) : void
observes
«interface» InventoryObserver
+onLowStock(productId, qty) : void
Singleton Pattern
«singleton» UserManager
-Map<String, User Account> users
-UserManager()
+static getInstance() : UserManager
+authenticate(username, password) : UserAccount
+createAccount(user) : void
+resetPassword(username, answer, newPassword) : boolean
Singleton Pattern
«singleton» DataSource
-static final DataSource INSTANCE
-Path dataDir
-DataSource()
+static getInstance() : DataSource
+getPath(filename) : Path


Previous UML Diagram
«interface»
UIScreen
+show(stage) : void
navigates to
navigates to
LoginPage
-Stage primaryStage
-UserManager userManager
+start(stage) : void
-createLoginCard() : VBox
-handleLogin() : void
-handleSignUp() : void
-handleForgotPassword() : void
-showAdminDashboard(user) : void
-showCustomerDashboard(user) : void
CustomerDashboard
-Stage primaryStage
-UserAccount currentUser
-BackendFacade backend
-ShoppingCart cart
+start(stage) : void
-createDashboardView() : Scene
-showShopView() : void
-showOrdersView() : void
-showProfileView() : void
-handleAddToCart(candy) : void
-handleCheckout() : void
-handleLogout() : void
uses
executes
manages
places orders
StatusBadge
+create(status) : Label
-getColor(status) : String
Command Pattern
«interface»
CartCommand
+execute() : void
+getName() : String
+getPrice() : double
+getDescription() : String
RemoveItemCommand
-ShoppingCart cart
-Candy item
-int quantity
+execute() : void
+getName() : String
+getPrice() : double
AddItemCommand
-ShoppingCart cart
-Candy item
-CandyPackage candyPackage
+execute() : void
+getName() : String
+getPrice() : double
operates on
Decorator Pattern
ShoppingCart
-List<Candy> items
-List<CandyPackage> packages
+addItem(candy, pkg) : void
+removeItem(candy) : void
+getItems() : List<Candy>
+clear() : void
operates on
«interface»
CandyPackage
+pack(candy) : void
+getPrice() : double
encapsulates
encapsulates
encapsulates
HardCandyBag
-double price
+pack(candy) : void
+getPrice() : double
GummyBag
-double price
+pack(candy) : void
+getPrice() : double
ChocolateBox
-double price
+pack(candy) : void
+getPrice() : double
BasicCandyPackage
-double price
+pack(candy) : void
+getPrice() : double
decorates
contains
«interface»
Candy
+getName() : String
+getPrice() : double
+getQuantity() : int
+setPrice(price) : void
+setQuantity(quantity) : void
Factory Pattern
«interface»
CandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
HardCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
ChocolateCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
GummyCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
BaseCandyFactory
+createCandy() : Candy
+createCandy(price) : Candy
+createCandy(price, quantity) : Candy
+createPackage() : CandyPackage
+createCandy(original) : Candy
creates
HardCandy
-String name
-double price
-int quantity
-boolean sugarFree
+isSugarFree() : boolean
ChocolateCandy
-String name
-double price
-int quantity
-String cocoaPercent
+getCocoaPercent() : String
GummyCandy
-String name
-double price
-int quantity
-String flavor
+getFlavor() : String
BaseCandy
-String name
-double price
-int quantity
+getName() : String
+getPrice() : double
+getQuantity() : int
+setPrice(price) : void
+setQuantity(quantity) : void
creates
creates
creates
creates
Console
+displayWelcome() : void
+displayMenu() : void
+getInput() : String
displays with
CandyStoreApp
-CandyStore store
-ShoppingCart cart
-Checkout checkout
+main(args) : void
+displayMenu() : void
+browseCandy() : void
+viewCart() : void
+checkout() : void
-getFactoryForCandy(candy) : CandyFactory
selects
«singleton»
Inventory
-static Inventory instance
-Map<String, Integer> stock
-Inventory()
+static getInstance() : Inventory
+getStock(productId) : int
+setStock(productId, qty) : void
Command Pattern
Checkout
-ShoppingCart cart
-PricingCommand pricingCommand
+Checkout(cart, pricingCommand)
+calculateTotal() : double
+process() : Order
uses
«interface»
PricingCommand
+apply(items) : double
DiscountPricingCommand
-double percentOff
+apply(items) : double
RegPricingCommand
+apply(items) : double
implements
implements
Order
-String orderId
-String customerId
-List<OrderItem> items
-double subtotal
-double discount
-OrderStatus status
-LocalDateTime createdAt
+addDiscount(amount) : void
+getTotal() : double
+setStatus(status) : void
belongs to
OrderItem
-String productId
-String productName
-int quantity
-double price
+getTotal() : double
contains
Customer
-String customerId
-String name
-String email
-String phone
+getCustomerId() : String
+getName() : String
persists
«singleton»
CandyStore
-static CandyStore instance
-List<Candy> catalog
-CandyStore()
+static getInstance() : CandyStore
+getCatalog() : List<Candy>
+addCandy(candy) : void
-loadInventoryFromCSV(filename) : void
persists
CustomerDataManager
-Path customersFile
+save(customer) : void
+findById(customerId) : Customer
+findAll() : List<Customer>
Command Pattern
BackendFacade
-InventoryService inventoryService
-OrderService orderService
-OrderDataManager orderDataManager
+checkout(cart, customer, discounts) : Order
+getInventory(Map)(productId) : int
+getAllOrders() : List<Order>
+getLowStockItems(threshold) : List
manages inventory
applies
«interface»
DiscountCommand
+name() : String
+apply(order) : void
PercentOff
-double percent
+name() : String
+apply(order) : void
CouponCodeCommand
-String code
-double flatAmount
+name() : String
+apply(order) : void
uses
OrderService
-OrderDataManager dataManager
+createOrder(customerId, items) : Order
+applyDiscounts(order, commands) : void
+updateOrderStatus(orderId, status) : void
+getOrdersByCustomer(customerId) : List<Order>
-decrementInventoryFor(items) : void
displays for
updates inventory
saves orders
UserAccount
-String username
-String passwordHash
-String role
-String email
-String securityQuestion
-String securityAnswer
+getRole() : String
+isAdmin() : boolean
uses
OrderDataManager
-Path ordersFile
-Path orderItemsFile
+save(order) : void
+load(id) : List<Order>
+findByCustomerId(customerId) : List<Order>
+updateStatus(orderId, status) : void
-loadOrders() : List<Order>
saves
Singleton Pattern
«singleton»
DataSource
-static final DataSource INSTANCE
-Path dataDir
-DataSource()
+static getInstance() : DataSource
+getPath(filename) : Path
facilities via subject
AdminDashboard
-Stage primaryStage
-UserAccount currentUser
-DashboardStatsManager statsManager
-BackendFacade backend
+start(stage) : void
-createHeader() : HBox
-createStatsCards() : HBox
-createOrdersTab() : Tab
-createInventoryTab() : Tab
-createUsersTab() : Tab
-handleLogout() : void
queries stats
StatsCard
-String title
-String value
-String icon
+StatsCard(title, value, icon)
+getCard() : VBox
OrderTableView
+createTable() : TableView<Order>
+populateTable(orders) : void
DashboardStatsManager
-OrderDataManager orderDataManager
-InventoryService inventoryService
-UserManager userManager
+getTotalInventoryValue() : double
+getLowStockCount() : int
+getTotalOrders() : int
+getTotalUsers() : int
+getRecentOrders(limit) : List<Order>
queries
Observer Pattern
InventoryService
-Map<String, Integer> stock
-InventorySubject subject
-int lowStockThreshold
+getStock(productId) : int
+setStock(productId, qty) : void
+decrement(productId, qty) : void
+onLowStock(observer) : void
manages
contains
InventorySubject
-List<InventoryObserver> observers
+attach(observer) : void
+detach(observer) : void
+notifyLowStock(productId, qty) : void
observes
«interface»
InventoryObserver
+onLowStock(productId, qty) : void
Singleton Pattern
«singleton»
UserManager
-static UserManager instance
-Map<String, UserAccount> users
-UserManager()
+static getInstance() : UserManager
+authenticate(username, password) : UserAccount
+createAccount(user) : void
+resetPassword(username, answer, newPassword) : boolean
authenticates
queries
queries
queries
updates inventory
saves orders
manages
contains
queries
persists
Contains

SECTION 3 - Third-party VS. Original Code

All project code is original. ChatGPT was used for guidance and debugging, particularly for UI development; usage is documented in code comments.

Third-party Resources utilized

- JavaFX Reference Docs when developing UI for available APIs and connecting CSS/ styling elements - openjfx.io
- Additional Reference for JavaFX applications: https://www.geeksforgeeks.org/java/javafx-tutorial/
- Maven Documentation: https://maven.apache.org/guides/getting-started/index.html
- Java Documentation for API/Libraries available: https://docs.oracle.com/en/java/
- MDN Documentation (CSS styling reference) - https://developer.mozilla.org/en-US/docs/Web/CSS

SECTION 4 - OOAD Process Statement

1. *Integration and Source Control Challenges:*

   During development, when trying to merge new work back to the main branch, individual work on components conflicted in source control, requiring cleanup and merge resolution.

2. *Connecting Front-end and Back-end services together:*

   Designing a way for the Back-end elements to communicate and provide/receive information from the front-end UI elements. Ultimately, we ended up with a Façade implementation to connect the two.

3. *Architecture Evolution:*

   The primary architecture shifted from differentiating online vs. in-person stores to a unified system focused on Customer/Admin roles. The differentiation between roles seemed to have more broad applicability and allowed us to think of how the different personas for the application would use it.