09 Breaking Bread

Elena Ingraham
Friedrich Amouzou
Hussain Quadri
Phi Trang

Vision

Our vision is to practice object oriented programming by designing a shop that utilizes a lot of the principles and patterns that we have learned in class.

Project description

Breaking Bread is an online bakery store with a catalogue of bakery utensils, merchandise, and customizable baked goods that are shipped directly to your home.

Summary

In the last two weeks we have built upon all of the classes we wrote before the last milestone. Namely we have completed fully functional versions of the Cart, Inventory and User Class. In order to test the implementation of all of our programs, we created a blob class that serves as a driver program. We have continued to work on the user interface. However, after learning that we will not be graded based on aesthetics it has subsequently been deprioritized. As per the requirements, we have applied the Singleton design pattern to the inventory class as well as the Strategy design pattern to the Deal class. Although the database will not factor in to the grading score, we thought it would be beneficial to learn how to create and connect a SQL database to a Java program. We have begun this progress and will complete it if time allows.

Breakdown

Friedrich Amouzou

- Made use-case diagram and documents
- Made class diagrams for both part 2 and 3
- Refactored and Implemented the deal system and wrote the Deal, FixedDeal, PercentDeal, and DateRange classes
- Implemented most of the non-custom Product class
- Created example login page in temp Application class
- Even more Java swing research

Hussain Quadri

- Created the sequence diagrams for part 2.
- Created, refactored the User class, as well as Admin and Customer subclasses. Updated their functionality for part 4.

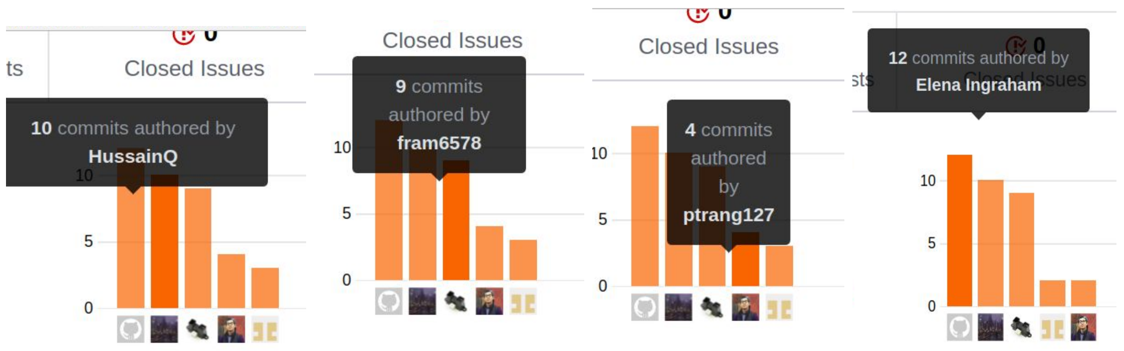
Phi Trang

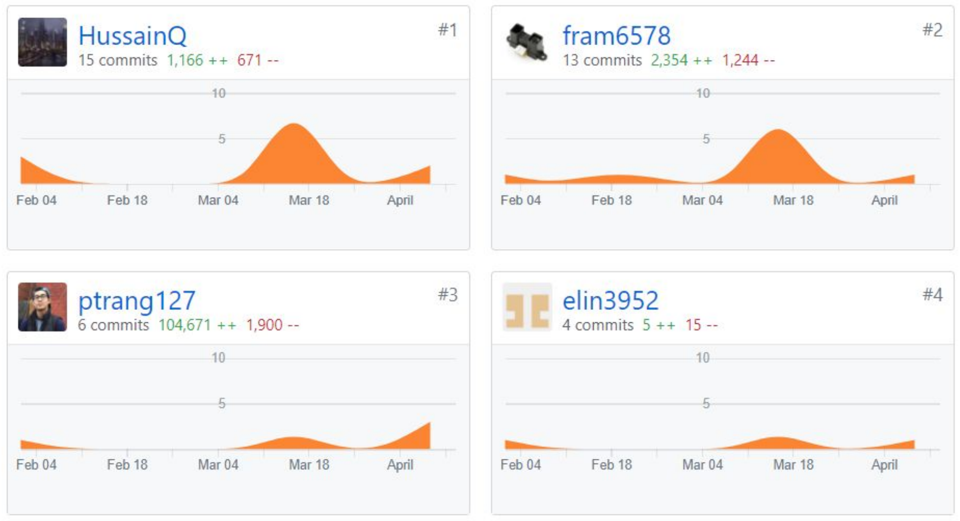
- Created Inventory skeleton class. Updated Cart class and Customer class
- Created a temporary blob.java class to use as a driver program
- Created functioning relationship between the inventory, cart, and customer
- Requirements analysis
- Overambitious mock UI
- Java Swing research

Elena Ingraham (Note Github graph is still inaccurate)

- Helped establish the Use Cases used in the Activity and Sequence diagrams
- Made the Activity Diagram in Part 2
- Worked on the Database, in hopes to have it fully connected for the completed project.
- Worked on the Cart class
- Implemented the Strategy design pattern on the Deal class

Github Graphs





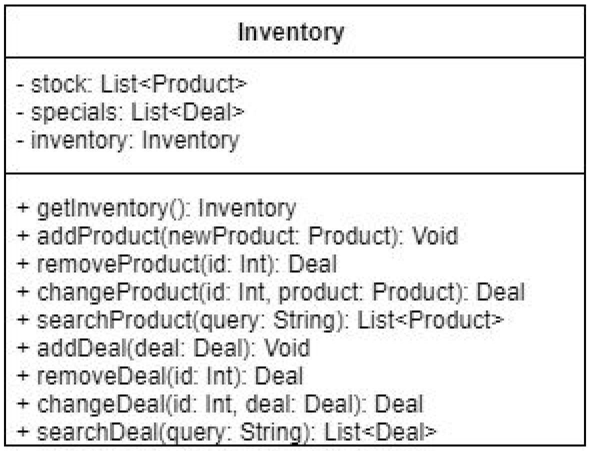
Estimate Remaining Effort

In terms of hours, the remaining effort will be less than the previous iteration since we are much more familiar with Java as well as the principles of OOD. As of right now, some of the programs are bare bones and strung together to prioritize functionality over OOD principles. Our work will be complete when we refactor our Blob class into a concise driver program. As of right now the Blob class can add products to the inventory list. It then creates a user who then searches through the inventory. If the item the dummy user does in fact exist within the inventory that item is returned. This incorporates and connects almost all of the parts involved in the project. Hopefully, time will allow for us to design a sleek user interface as well as a connected database.

Design Patterns

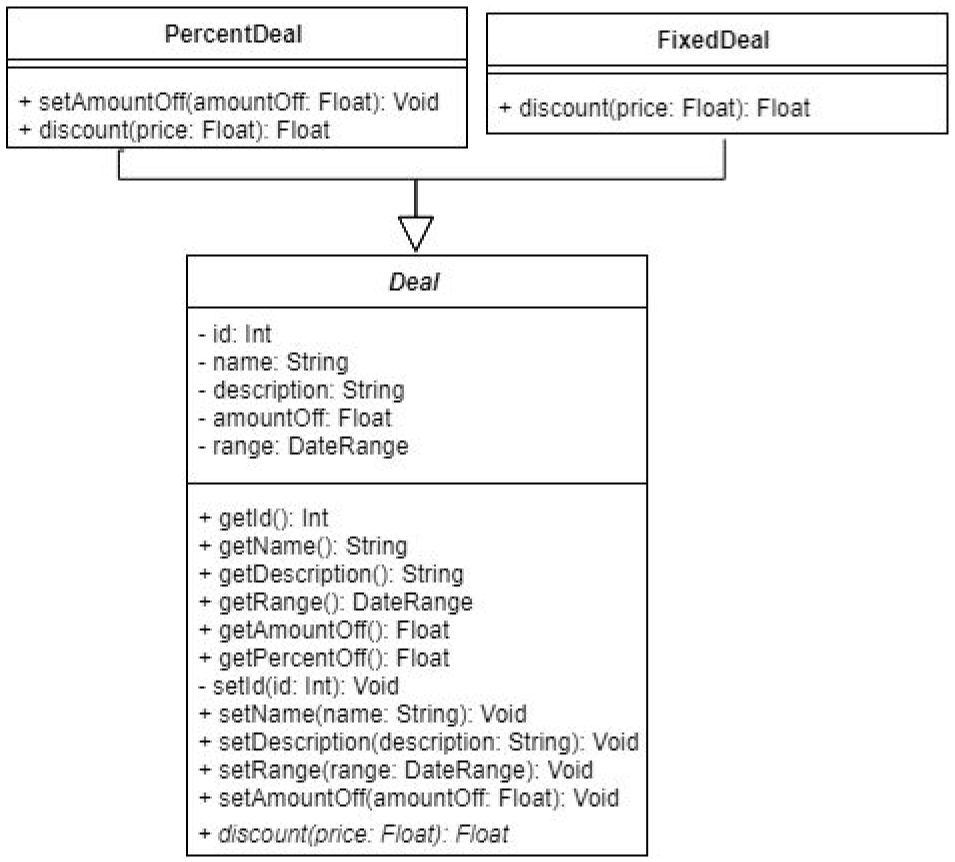
Description

Singleton : Although it is the most frequently abused design pattern, we implemented our Inventory Class following the conventions of Singleton. Since only one Inventory() object will be instantiated within the Main class, this is an appropriate use of Singleton and does not violate any OOD principles.



Inventory
- stock: List<Product>
- specials: List<Deal>
- inventory: Inventory
+ getInventory(): Inventory
+ addProduct(newProduct: Product): Void
+ removeProduct(id: Int): Deal
+ changeProduct(id: Int, product: Product): Deal
+ searchProduct(query: String): List<Product>
+ addDeal(deal: Deal): Void
+ removeDeal(id: Int): Deal
+ changeDeal(id: Int, deal: Deal): Deal
+ searchDeal(query: String): List<Deal>

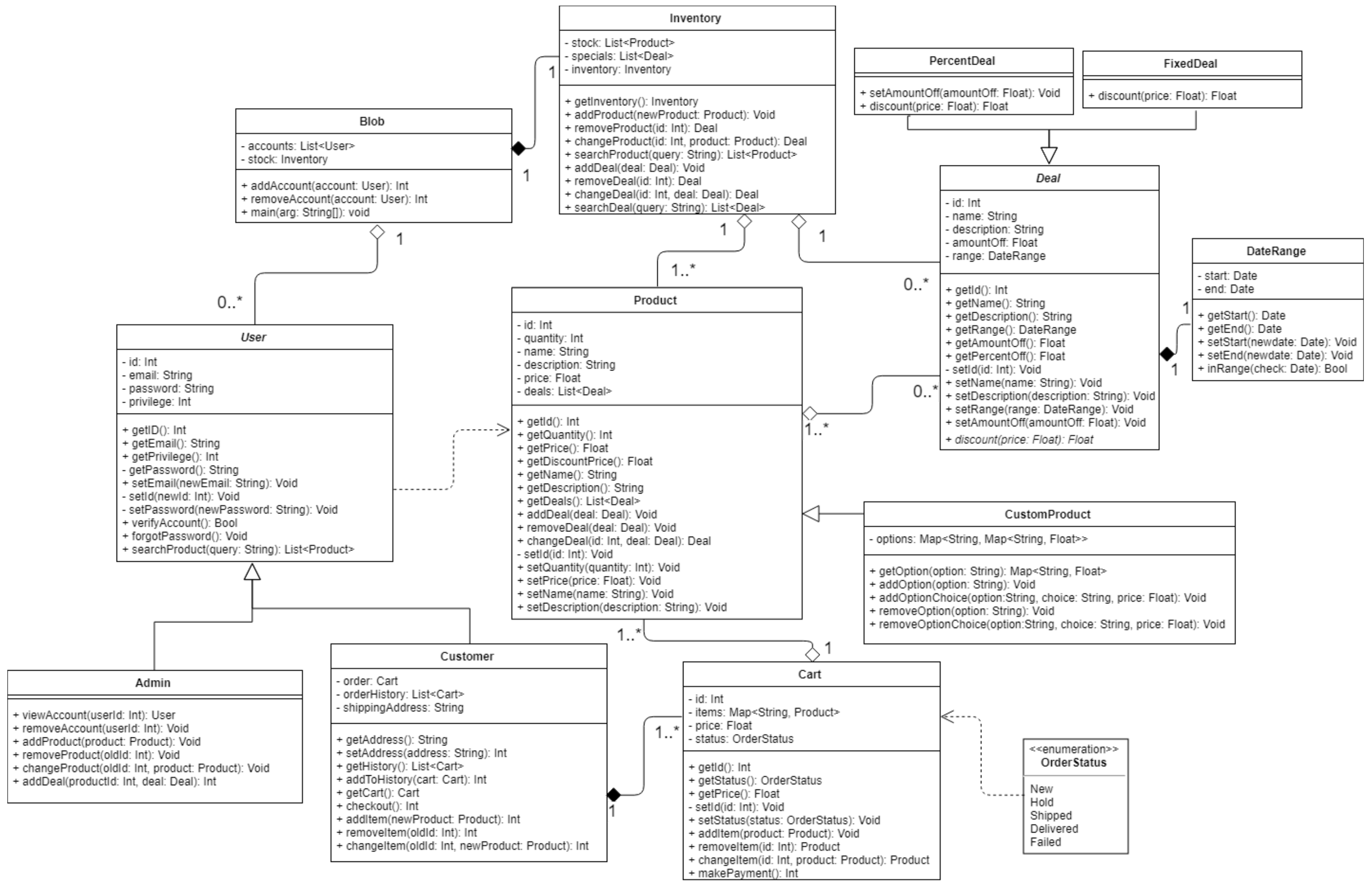
Strategy: Since there are different discounts being applied the Strategy design pattern is helpful for easily making use of multiple algorithms. In our case the algorithms are various discount amounts which are applied to products in the Cart. By using the Strategy design pattern we minimize coupling and program to an interface as is prefered. The pattern also allow for the system to better follow the Open Closed principle of OOD since the client will not have to re-tinker with the Cart class if the discounts change or expire.


PercentDeal
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float
FixedDeal
+ discount(price: Float): Float
Deal
- id: Int
- name: String
- description: String
- amountOff: Float
- range: DateRange
+ getId(): Int
+ getName(): String
+ getDescription(): String
+ getRange(): DateRange
+ getAmountOff(): Float
+ getPercentOff(): Float
- setId(id: Int): Void
+ setName(name: String): Void
+ setDescription(description: String): Void
+ setRange(range: DateRange): Void
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float

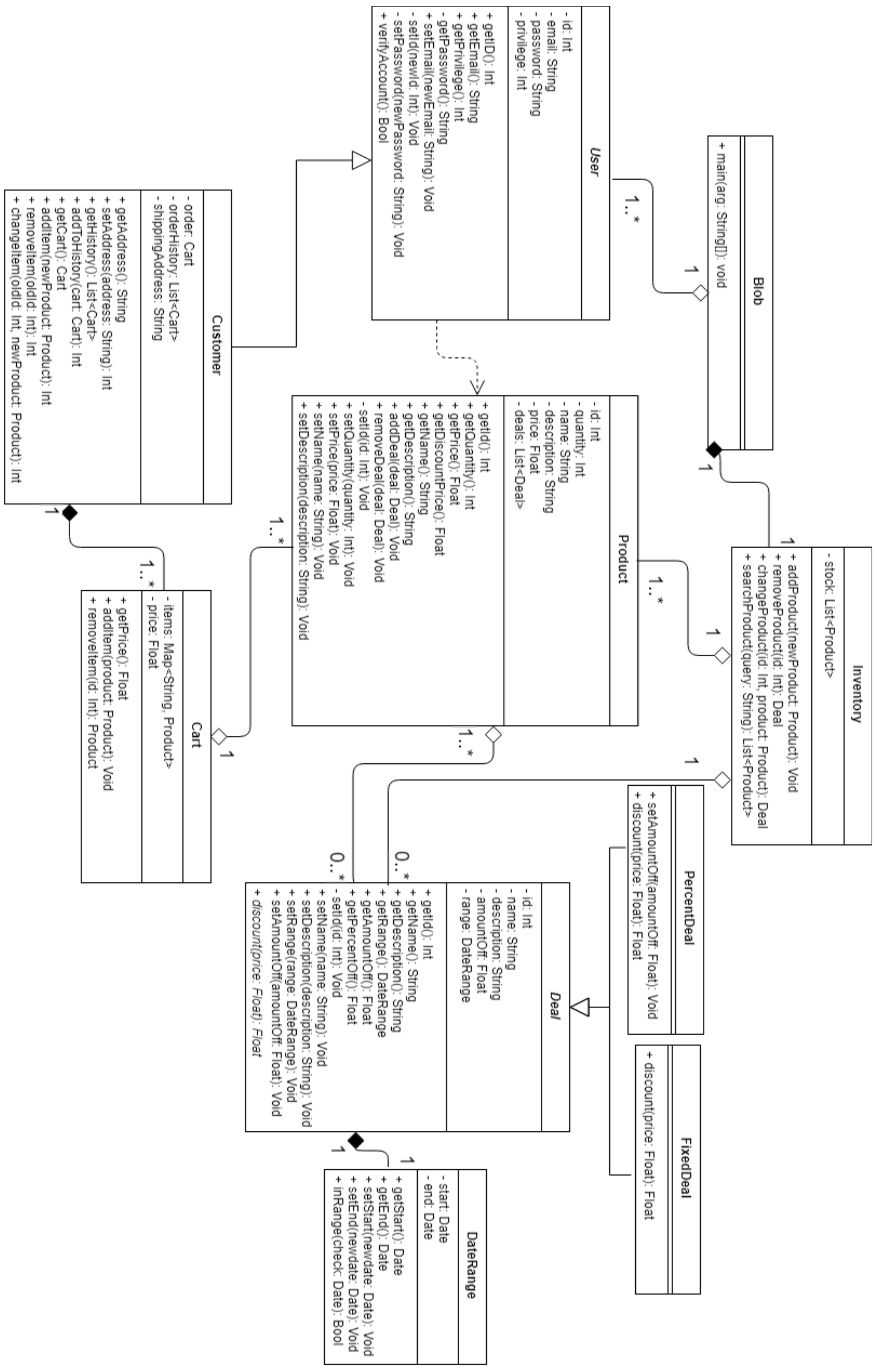
Final Iteration

For our final iteration we will clean up the existing code. After continuing to comprehensively work through test cases we will transform our blob class into a driver program. We will make sure users can seamlessly navigate all use cases described in in the second submission. While cleaning up the existing code, we will look for code smells and apply refactoring techniques where necessary. We will also complete the user interface to the best of our abilities for as long as time allows. The database will be completed and connected before the final submission.

Class diagram


Inventory
- stock: List<Product>
- specials: List<Deal>
- inventory: Inventory
+ getInventory(): Inventory
+ addProduct(newProduct: Product): Void
+ removeProduct(id: Int): Deal
+ changeProduct(id: Int, product: Product): Deal
+ searchProduct(query: String): List<Product>
+ addDeal(deal: Deal): Void
+ removeDeal(id: Int): Deal
+ changeDeal(id: Int, deal: Deal): Deal
+ searchDeal(query: String): List<Deal>
PercentDeal
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float
FixedDeal
+ discount(price: Float): Float
Blob
- accounts: List<User>
- stock: Inventory
+ addAccount(account: User): Int
+ removeAccount(account: User): Int
+ main(arg: String[]): void
Deal
- id: Int
- name: String
- description: String
- amountOff: Float
- range: DateRange
+ getId(): Int
+ getName(): String
+ getDescription(): String
+ getRange(): DateRange
+ getAmountOff(): Float
+ getPercentOff(): Float
- setId(id: Int): Void
+ setName(name: String): Void
+ setDescription(description: String): Void
+ setRange(range: DateRange): Void
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float
DateRange
- start: Date
- end: Date
+ getStart(): Date
+ getEnd(): Date
+ setStart(newdate: Date): Void
+ setEnd(newdate: Date): Void
+ inRange(check: Date): Bool
User
- id: Int
- email: String
- password: String
- privilege: Int
+ getID(): Int
+ getEmail(): String
+ getPrivilege(): Int
- getPassword(): String
+ setEmail(newEmail: String): Void
- setId(newId: Int): Void
- setPassword(newPassword: String): Void
+ verifyAccount(): Bool
+ forgotPassword(): Void
+ searchProduct(query: String): List<Product>
Product
- id: Int
- quantity: Int
- name: String
- description: String
- price: Float
- deals: List<Deal>
+ getId(): Int
+ getQuantity(): Int
+ getPrice(): Float
+ getDiscountPrice(): Float
+ getName(): String
+ getDescription(): String
+ getDeals(): List<Deal>
+ addDeal(deal: Deal): Void
+ removeDeal(deal: Deal): Void
+ changeDeal(id: Int, deal: Deal): Deal
- setId(id: Int): Void
+ setQuantity(quantity: Int): Void
+ setPrice(price: Float): Void
+ setName(name: String): Void
+ setDescription(description: String): Void
CustomProduct
- options: Map<String, Map<String, Float>>
+ getOption(option: String): Map<String, Float>
+ addOption(option: String): Void
+ addOptionChoice(option:String, choice: String, price: Float): Void
+ removeOption(option: String): Void
+ removeOptionChoice(option:String, choice: String, price: Float): Void
Admin
+ viewAccount(userId: Int): User
+ removeAccount(userId: Int): Void
+ addProduct(product: Product): Void
+ removeProduct(oldId: Int): Void
+ changeProduct(oldId: Int, product: Product): Void
+ addDeal(productId: Int, deal: Deal): Int
Customer
- order: Cart
- orderHistory: List<Cart>
- shippingAddress: String
+ getAddress(): String
+ setAddress(address: String): Int
+ getHistory(): List<Cart>
+ addToHistory(cart: Cart): Int
+ getCart(): Cart
+ checkout(): Int
+ addItem(newProduct: Product): Int
+ removeItem(oldId: Int): Int
+ changeItem(oldId: Int, newProduct: Product): Int
Cart
- id: Int
- items: Map<String, Product>
- price: Float
- status: OrderStatus
+ getId(): Int
+ getStatus(): OrderStatus
+ getPrice(): Float
- setId(id: Int): Void
+ setStatus(status: OrderStatus): Void
+ addItem(product: Product): Void
+ removeItem(id: Int): Product
+ changeItem(id: Int, product: Product): Product
+ makePayment(): Int
<<enumeration>>
OrderStatus
New
Hold
Shipped
Delivered
Failed
1
1
1
0..*
1
1
1..*
0..*
1
1
0..*
1..*
1..*
1
1..*
1

Completed Class diagram


User
- id: Int
- email: String
- password: String
- privilege: Int
+ getID(): Int
+ getEmail(): String
+ getPrivilege(): Int
- getPassword(): String
+ setEmail(newEmail: String): Void
- setId(newId: Int): Void
- setPassword(newPassword: String): Void
+ verifyAccount(): Bool
Blob
+ main(arg: String[]): void
Customer
- order: Cart
- orderHistory: List<Cart>
- shippingAddress: String
+ getAddress(): String
+ setAddress(address: String): Int
+ getHistory(): List<Cart>
+ addToHistory(cart: Cart): Int
+ getCart(): Cart
+ addItem(newProduct: Product): Int
+ removeItem(oldId: Int): Int
+ changeItem(oldId: Int, newProduct: Product): Int
Product
- id: Int
- quantity: Int
- name: String
- description: String
- price: Float
- deals: List<Deal>
+ getId(): Int
+ getQuantity(): Int
+ getPrice(): Float
+ getDiscountPrice(): Float
+ getName(): String
+ getDescription(): String
+ addDeal(deal: Deal): Void
+ removeDeal(deal: Deal): Void
- setId(id: Int): Void
+ setQuantity(quantity: Int): Void
+ setPrice(price: Float): Void
+ setName(name: String): Void
+ setDescription(description: String): Void
Inventory
- stock: List<Product>
+ addProduct(newProduct: Product): Void
+ removeProduct(id: Int): Deal
+ changeProduct(id: Int, product: Product): Deal
+ searchProduct(query: String): List<Product>
Cart
- items: Map<String, Product>
- price: Float
+ getPrice(): Float
+ addItem(product: Product): Void
+ removeItem(id: Int): Product
Deal
- id: Int
- name: String
- description: String
- amountOff: Float
- range: DateRange
+ getId(): Int
+ getName(): String
+ getDescription(): String
+ getRange(): DateRange
+ getAmountOff(): Float
+ getPercentOff(): Float
- setId(id: Int): Void
+ setName(name: String): Void
+ setDescription(description: String): Void
+ setRange(range: DateRange): Void
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float
PercentDeal
+ setAmountOff(amountOff: Float): Void
+ discount(price: Float): Float
FixedDeal
+ discount(price: Float): Float
DateRange
- start: Date
- end: Date
+ getStart(): Date
+ getEnd(): Date
+ setStart(newdate: Date): Void
+ setEnd(newdate: Date): Void
+ inRange(check: Date): Bool
1..*
1
1
1
1..*
1
1
1..*
1
1..*
1
1..*
1
0..*
0..*
1
1