Sales System Detailed Documentation

This document explains the purpose of each class (Model) in the system, detailing the internal logic and how objects interact to complete the full sales and invoicing lifecycle. The project is built on the principles of Object-Oriented Programming (OOP) to ensure modularity and maintainability.

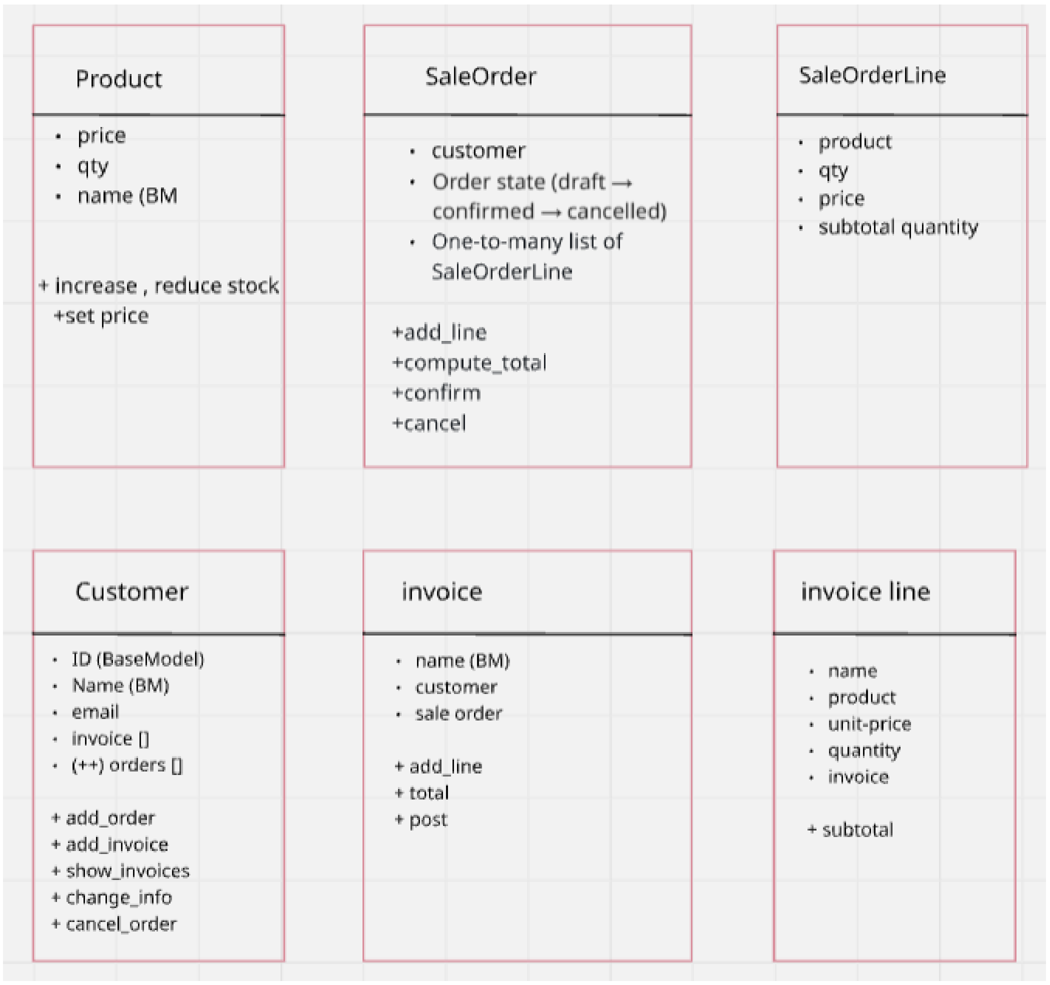
1. Project Architecture and Workflow

The system is designed around a clear hierarchical structure, where key transaction documents (SaleOrder, Invoice) are composed of line items (SaleOrderLine, InvoiceLine).

A. Core Relationships

Entity	**Purpose**	**Key Relationships**
BaseModel	Abstract Base Class (ABC) providing shared properties like unique id and name.	Parent class for all business models.
Customer	The buyer entity, tracking contact details and transactional history.	Has-many Invoices and SaleOrders.
Product	The item for sale, managing price and crucial inventory quantity.	Referenced by order/invoice lines.
SaleOrder	The main transaction document, controlling the order state (Draft/Confirmed/Cancelled).	Linked to a Customer. Contains SaleOrderLines.
Invoice	The financial document automatically created upon order confirmation.	Linked to a Customer and a SaleOrder. Contains InvoiceLines.

UML Class Diagram



B. The Sales Workflow

The core business process is centralized in the SaleOrder.confirm() method:

1. **Creation:** A SaleOrder is initiated for a specific Customer (starting in 'draft' state).
2. **Confirmation (Critical Step):** Calling SaleOrder.confirm() executes the financial and inventory transactions:
   - **Inventory Reduction:** Stock is reduced from Products.
   - **Invoice Generation:** A new Invoice is instantiated, its lines are populated, and the invoice is immediately set to the final 'posted' state.
   - **Customer Record Update:** The newly posted invoice is added to the Customer's permanent list of invoices.
3. **Cancellation:** SaleOrder.cancel() reverses the process by restoring stock and updating the order state.

2. All Model and Function Documentation

This section provides a detailed breakdown of all classes, their attributes, and their public methods.

2.1. BaseModel Class (File: base.py)

This class provides a common base for all other models, ensuring every object has a unique identifier and a name.

Attribute / Method	**Description**
id	A globally unique identifier (UUID) generated upon instantiation.
name	The name or reference of the object (e.g., Customer Name, Invoice ID).
__init__(self, name)	Initializes the object with a name and generates the unique id.

2.2. Customer Class (File: customer.py)

Models the buyer. Manages customer details and aggregates their historical transactions.

Function / Property	**Parameters**	**Purpose and Explanation**
__init__(self, name, email)	name: str, email: str	Initializes the customer, setting up email and creating empty lists for transactional history (self.invoices, self.orders).
add_order(self, order)	order: SaleOrder	Adds a SaleOrder object to the customer's open orders list.
add_invoice(self, invoice)	invoice: Invoice	**Core Requirement.** Adds a confirmed Invoice object

		to the customer's permanent record (self.invoices).
show_invoices(self)	None	**Core Requirement.** Iterates through self.invoices and prints a formatted summary for each (Name, Total, State).
cancel_order(self, order)	order: SaleOrder	Removes a draft order from the customer's orders list.

2.3. Product Class (File: product.py)

Models an item of inventory. Crucial for stock management.

Function / Property	**Parameters**	**Purpose and Explanation**
__init__(self, name, price, quantity)	name: str, price: float, quantity: int	Initializes product with price and starting inventory quantity.
reduce_stock(qty)	qty: int	Decreases stock; includes validation to prevent negative inventory. Used upon order confirmation.
increase_stock(qty)	qty: int	Increases stock; used when a confirmed order is cancelled.
set_price(new_price)	new_price: float	Updates the product's selling price.

2.4. SaleOrder Class (File: sale_order.py)

The primary document capturing the customer's intent to purchase.

Function / Property	Parameters	Purpose and Explanation
__init__(self, name, customer)	name: str, customer: Customer	Initializes the order, linking it to the customer. State starts as "draft".
add_line(self, product, quantity)	product: Product, quantity: int	Creates and adds a SaleOrderLine. Includes checks for stock availability.
compute_total()	None	Calculates the total financial value of the order.
confirm()	None	**Triggers Inventory/Financial Flow.** Reduces stock, creates and posts the Invoice, and updates the order state to "confirmed".
cancel()	None	Reverts a confirmed order. Restores product stock to inventory and sets state to "cancelled".

2.5. SaleOrderLine Class (File: sale_order_line.py)

A detail model linking a SaleOrder to a specific Product and quantity.

Function / Property	Parameters	Purpose and Explanation
__init__(self, product, quantity)	product: Product, quantity: int	Initializes the line with the referenced product and ordered quantity.
subtotal (Property)	None	Calculates the total cost for this line: quantity * product.price.

2.6. Invoice Class (File: invoice.py)

The final, legally binding financial document.

Function / Property	Parameters	Purpose and Explanation
__init__(self, name, customer, sale_order)	name: str, customer: Customer, sale_order: SaleOrder	Initializes the invoice, linking it to the respective customer and sale_order. State is "draft".
add_line(self, product, quantity)	product: Product, quantity: int	Creates a new InvoiceLine and adds it to the invoice. Prevents adding lines to a "posted" invoice.
total (Property)	None	A computed property that returns the grand total of the invoice by summing the subtotal of all its lines.
post(self)	None	**Finalizes the Invoice.** Changes the state from "draft" to "posted".

2.7. InvoiceLine Class (File: invoice_line.py)

A detail model tracking the specific products and charged amounts on the Invoice.

Function / Property	Parameters	Purpose and Explanation
__init__(self, name, product, quantity, unit_price, invoice)	Various	Initializes the line with the product reference, quantity, unit price (fixed at the time of invoicing), and the parent invoice.
subtotal (Property)	None	A computed property calculating the total cost

		for this line: self.quantity * self.unit_price.

3. Main Application Logic (main.py)

These functions manage user interaction and orchestrate the calls to the object methods.

1. Add New Customer (Main Menu Option 2)

Description	**Logic in main.py**
Functionality	**Handles the process of creating a new customer.** Prompts the user for the required name and email.
Action	A new Customer object is instantiated (Customer(name, email)) and then permanently added to the global customers list.

2. Show Customer Invoices (Main Menu Option 5)

Description	**Logic in main.py**
User Interaction	Displays a numbered list of all existing customers and prompts the user to select one by index.
Action	**Delegates responsibility.** Retrieves the selected Customer object from the global list and calls the method: customer.show_invoices(). The customer object is responsible for displaying its own financial history.

3. Other Utility Functions

Function Name	Purpose
list_products(products)	Prints a formatted list of all available products and their current stock levels.
list_customers(customers)	Prints a numbered list of all customers for selection.
create_sale_order(...)	A comprehensive function that handles the selection of a customer, adding/removing products, and confirming or saving the order.
cancel_order(orders)	Displays confirmed orders and allows the user to select one for cancellation, triggering the stock restoration.

4. Design Choice:

· Menu-driven interface keeps user interactions simple

· Functions separated for modularity and easier debugging

· Adding new customers integrated into menu

5. Project Workflow

1. Start program → Main menu appears.
2. Create Sale Order → Select customer → Add products → Confirm.
3. Confirmed orders update stock and generate invoices automatically.
4. Orders can be canceled from main menu; stock is restored.
5. View invoices per customer as needed.
6. Add new customers dynamically.

6. Design Principles

· **Modularity:** Each class/function has a single responsibility.

· **Encapsulation:** Stock and order logic encapsulated to avoid invalid states.

· **State Management:** Prevents invalid operations on confirmed/canceled orders.

· **Extensibility:** Can easily add discounts, multiple currencies, or shipments.

Comments and Logging

· Each function contains **comments** describing purpose, validation, and side effects.

· [INFO] logs provide clear runtime feedback and help debug the system.

Link for the repository on GitHub:

https://github.com/esraasoffar/sales-module-implementation-project