Pseudocode for both files/classes included in one file for brevity. Header files are implied, as all attributes/functions/parameters/etc are already shown.

Classes based on diagram:

Start of Customer.cpp

```
Class Customer
    Attributes:
        Public:
            integer customerID
            string name
            boolean credit          //credit denotes if they are able to make a purchase with insufficient funds in balance
            double balancedollars
            vector of Products productsPurchased
            Private:
            No private attributes needed
    End of Attributes for Customer
    Functions:
        //initialize Customer with ID, name, and if customer uses credit
        Customer(customerID, name, credit){
            throw runtime error if name is an empty string
            set ID, name, and credit values of CUstomer object to input values
        }

        integer getID(){
            return current Customer's ID
        }

        string getName(){
            return current Customer's name
        }

        void setName(take in string containing new name){
            throw runtime error if name is an empty string
            set current Customer's name to the new name provided
        }

        boolean getCredit(){
            return current Customer's credit status
        }
```

```
                void setCredit(boolean for new credit status){
                        set current Customer's credit status to the input status
                }

                integer getBalance(){
                        return current CUstomer's current balance
                }

                void processPayment(double amount){
                        throw runtime error if amount is negative
                        add the amount to the Customer's balance
                }

                void processPurchase(double amount, Product product){
                        throw a runtime error if amount is negative
                        if credit is false, throw a runtime error if the price is greater than the
Customer's balance
                        if credit is true, allow user to purchase even if balance would become
negative
                        subtracts payment form user's balance
                        if not already on list, adds the given product to the user's list of purchased
products
                }
        End of Functions for Customer
End of Customer Class

std::ostream& operator<<(std::ostream& os, const Customer& c){
        os << c.to_str();
        return os;
}


End of Customer.cpp

Start of Product.cpp

Class Product
        Attributes:
                Public:
                        integer productID
                        string name
                        string description
                Private:
```

integer inventory // amount of item in stock
integer numSold
double totalPaid // total amount of money paid for this item.
End of Attributes for Product
Functions:
Product Product(integer productID, string productName){
throw runtime error if name is an empty string
initialize product object with input values as attributes
}

integer getID(){
returns ID of current object
}

string getName(){
returns ID of current object
}

void setName(string newName){
throw runtime error if name is an empty string
sets name of current object to new name
}

string getDescription(){
returns description of current object
}

void setName(string newDecription){
sets decription of current object to new name
}

integer getNumberSold(){
returns number of current objects that have been sold
}

double getTotalPaid(){
returns the total amount of money spent on getting shipments of this item
//make sure it works with addShipment()
}

integer getInventoryCount(){
returns amount of current object in inventory
}

```
		double getPrice(){
			returns current object's price in dollars, based on abg cost over time +
25% markup
			//ie price = (totalPaid / (inventory + numSold)) * 1.25
			//may need to format output to only show dollars and cents
			//avoid int/double division, use <double>wrappers or something
		}

		void addShipment(integer quantity, double cost){
			throw runtime error if either value is negative
			adds the entered qunatity of current item to inventory, and increases total
shipment cost of item by cost
		}

		void reduceInventory(integer quantity){
			throw runtime error if quantity is greater than amount of entity or if
quantity is negative
			removes the entered qunatity of current item from inventory
			increases number of items sold by quantity
		}
	End of Functions for Product
End of Product Class

std::ostream& operator<<(std::ostream& os, const Product& p){
	os << p.to_str();
	return os;
}
```