Codes for the program

1. inventory.py

```
# Constants for file names
PRODUCTS_FILE_NAME = "products.txt"  # File to store product data
SUPPLIERS_FILE_NAME = "suppliers.txt"  # File to store supplier data
ORDERS_FILE_NAME = "orders.txt"  # File to store order data

# --- PRODUCTS MANAGEMENT ---

# Function to load products from the file into a dictionary
def load_products():
    products = {}
    try:
        with open(PRODUCTS_FILE_NAME, "r") as file:  # Open the products
file in read mode
            for line in file:
                parts = line.strip().split("|")  # Split each line by the
delimiter "|"
                product_id = parts[0]  # Extract product ID
                name = parts[1]  # Extract product name
                price = float(parts[2])  # Convert price to float
                quantity = int(parts[3])  # Convert quantity to integer
                products[product_id] = {"name": name, "price": price,
"quantity": quantity}  # Add to the dictionary
    except FileNotFoundError:
        print("No products file found. Starting with an empty product
list.")  # Handle missing file gracefully
    return products

# Function to save the current state of products into the file
def save_products(products):
    with open(PRODUCTS_FILE_NAME, "w") as file:  # Open the products file
in write mode
        for product_id, details in products.items():
            # Write product details in the specified format

file.write(f"{product_id}|{details['name']}|{details['price']}|{details['qu
antity']}\n")

# Function to add a new product to the inventory
def add_new_product(products):
    product_id = input("Enter Product ID: ")  # Get unique product ID from
the user
    if product_id in products:  # Check if the product ID already exists
        print("This product ID already exists. Please try updating the
product instead.")
        return
```

```
    name = input("Enter Product Name: ")  # Get product name
    try:
        price = float(input("Enter Product Price: "))  # Get product price
as a float
        quantity = int(input("Enter Product Quantity: "))  # Get product
quantity as an integer
    except ValueError:
        # Handle invalid input for price or quantity
        print("Invalid price or quantity input. Please try again.")
        return
    # Add the new product to the dictionary
    products[product_id] = {"name": name, "price": price, "quantity":
quantity}
    save_products(products)  # Save updated products to the file
    print(f"Product '{name}' added successfully!")

# Function to update the details of an existing product
def update_product_details(products):
    product_id = input("Enter Product ID to update: ")  # Get the product
ID to update
    if product_id not in products:  # Check if the product exists
        print("Product not found. Please add it first.")
        return
    print("Current product details:")
    print(products[product_id])  # Display current details for reference
    # Allow user to update each detail or keep it the same
    name = input("Enter New Product Name (or press Enter to keep it the
same): ")
    price = input("Enter New Product Price (or press Enter to keep it the
same): ")
    quantity = input("Enter New Product Quantity (or press Enter to keep it
the same): ")
    if name:  # Update name if a new value is provided
        products[product_id]["name"] = name
    if price:  # Update price if a valid value is provided
        try:
            products[product_id]["price"] = float(price)
        except ValueError:
            print("Invalid price input. Keeping the old price.")
    if quantity:  # Update quantity if a valid value is provided
        try:
            products[product_id]["quantity"] = int(quantity)
        except ValueError:
            print("Invalid quantity input. Keeping the old quantity.")
    save_products(products)  # Save updated products to the file
    print("Product details updated successfully!")
```

```
# Function to display all products in the inventory
def display_products(products):
    print("Products Inventory")
    print("~" * 30) # Decorative separator for better readability.
    for product_id, details in products.items():
        # Display product details in a formatted manner
        name = details["name"]
        price = details["price"]
        quantity = details["quantity"]
        print(f"{product_id}: {name} - RM{price:.2f}, {quantity} units")
    print("~" * 30) # Decorative separator for better readability.

# --- SUPPLIERS MANAGEMENT ---

# Function to load suppliers from the file into a dictionary
def load_suppliers():
    suppliers = {}
    try:
        with open(SUPPLIERS_FILE_NAME, "r") as file:  # Open the suppliers
file in read mode
            for line in file:
                parts = line.strip().split("|")  # Split each line by the
delimiter "|"
                supplier_id = parts[0]  # Extract supplier ID
                name = parts[1]  # Extract supplier name
                contact_info = parts[2]  # Extract supplier contact info
                suppliers[supplier_id] = {"name": name, "contact_info":
contact_info}  # Add to the dictionary
    except FileNotFoundError:
        print("No suppliers file found. Starting with an empty supplier
list.")  # Handle missing file gracefully
    return suppliers

# Function to save the current state of suppliers into the file
def save_suppliers(suppliers):
    with open(SUPPLIERS_FILE_NAME, "w") as file:  # Open the suppliers file
in write mode
        for supplier_id, details in suppliers.items():
            # Write supplier details in the specified format

Fwrite(f"{supplier_id}|{details['name']}|{details['contact_info']}\n")

# Function to add a new supplier to the list
def add_new_supplier(suppliers):
    supplier_id = input("Enter Supplier ID: ")  # Get unique supplier ID
from the user
    if supplier_id in suppliers:  # Check if the supplier ID already exists
```

```
        print("This supplier ID already exists.")
        return
    name = input("Enter Supplier Name: ")  # Get supplier name
    contact_info = input("Enter Supplier Contact Information: ")  # Get
supplier contact info
    suppliers[supplier_id] = {"name": name, "contact_info": contact_info}
# Add to the dictionary
    save_suppliers(suppliers)  # Save updated suppliers to the file
    print(f"Supplier '{name}' added successfully!")

# Function to display all suppliers in the system
def display_suppliers(suppliers):
    print("Suppliers Information")
    print("~" * 30) # Decorative separator for better readability.
    for supplier_id, details in suppliers.items():
        # Display supplier details in a formatted manner
        name = details["name"]
        contact_info = details["contact_info"]
        print(f"{supplier_id}: {name} - Contact: {contact_info}")
    print("~" * 30) # Decorative separator for better readability.

# --- ORDERS MANAGEMENT ---

# Function to load orders from the file into a list
def load_orders():
    orders = []  # Initialize an empty list to store orders
    try:
        with open(ORDERS_FILE_NAME, "r") as file:  # Try to open the orders
file in read mode
            for line in file:  # Loop through each line in the file
                parts = line.strip().split("|")  # Remove trailing
spaces/newlines and split line by "|"
                order_id = parts[0]  # Extract the order ID (first field)
                product_id = parts[1]  # Extract the product ID (second
field)
                quantity = int(parts[2])  # Convert the quantity (third
field) to an integer
                supplier_id = parts[3]  # Extract the supplier ID (fourth
field)
                # Append the order details as a dictionary to the orders
list
                orders.append({"order_id": order_id, "product_id":
product_id, "quantity": quantity, "supplier_id": supplier_id})
    except FileNotFoundError:
        # If the file is not found, inform the user and start with an empty
list
        print("No orders file found. Starting with an empty order list.")
```

```
    return orders  # Return the list of orders

# Function to save orders to the file
def save_orders(orders):
    with open(ORDERS_FILE_NAME, "w") as file:  # Open the orders file in
write mode
        for order in orders:  # Loop through each order in the orders list
            # Write the order details to the file, separated by "|"

file.write(f"{order['order_id']}|{order['product_id']}|{order['quantity']}|
{order['supplier_id']}\n")

# Function to place a new order
def place_order(products, suppliers, orders):
    product_id = input("Enter the Product ID you want to order: ")  #
Prompt user to input Product ID
    if product_id not in products:  # Check if the product ID exists in the
products dictionary
        print("Product not found. Please add it first.")  # Inform the user
if the product is not found
        return  # Exit the function early

    supplier_id = input("Enter the Supplier ID: ")  # Prompt user to input
Supplier ID
    if supplier_id not in suppliers:  # Check if the supplier ID exists in
the suppliers dictionary
        print("Supplier not found. Please add the supplier first.")  #
Inform the user if the supplier is not found
        return  # Exit the function early

    try:
        # Prompt user to input the quantity they want to order and convert
it to an integer
        quantity = int(input(f"Enter the quantity of
{products[product_id]['name']} to order: "))
    except ValueError:
        # If the input is not a valid integer, inform the user and exit the
function
        print("Invalid quantity input. Please try again.")
        return

    # Check if the quantity requested exceeds the available stock for the
product
    if quantity > products[product_id]["quantity"]:
        print("Not enough stock available to fulfill this order.")  #
Inform the user if stock is insufficient
        return
```

```
    # Generate a new order ID by incrementing the current list size
    order_id = len(orders) + 1
    # Add the new order to the orders list as a dictionary
    orders.append({"order_id": str(order_id), "product_id": product_id,
"quantity": quantity, "supplier_id": supplier_id})
    # Deduct the ordered quantity from the product's available stock
    products[product_id]["quantity"] -= quantity
    save_orders(orders)  # Save the updated orders list to the file
    save_products(products)  # Save the updated product stock to the file
    # Confirm the successful placement of the order to the user
    print(f"Order placed successfully for {quantity} of
{products[product_id]['name']}!")


# --- REPORTING ---

# Function to generate a report of products with low stock levels.
def generate_low_stock_report(products, threshold=10):
    # Print the header for the Low Stock Report.
    print("Low Stock Report")
    print("~" * 30)  # Decorative separator for better readability.

    low_stock_found = False  # Flag to check if any low stock items exist.

    # Iterate over each product in the dictionary.
    for product_id, details in products.items():
        # Check if the product's quantity is less than or equal to the
threshold.
        if details["quantity"] <= threshold:
            # Print the product details if it has low stock.
            print(f"{details['name']} (ID: {product_id}) -
{details['quantity']} units left.")
            low_stock_found = True  # Set the flag to indicate low stock
was found.

    # If no products with low stock were found, notify the user.
    if not low_stock_found:
        print("No products are running low on stock.")

    print("~" * 30)  # Decorative separator for better readability.

# Function to generate a report of sales for each product.
def generate_sales_report(orders, products):
    # Print the header for the Sales Report.
    print("Sales Report")
    print("~" * 30)  # Decorative separator for better readability.
```

```
    sales = {}  # Dictionary to store the total sales for each product.

    # Iterate over each order in the list.
    for order in orders:
        product_id = order["product_id"]  # Get the product ID from the
order.

        # Add the quantity sold to the corresponding product ID in the
sales dictionary.
        if product_id in sales:
            sales[product_id] += order["quantity"]
        else:
            sales[product_id] = order["quantity"]

    # Iterate through the sales dictionary to generate the report.
    for product_id, quantity_sold in sales.items():
        # Print the product name, ID, and total units sold.
        print(f"Product {products[product_id]['name']} (ID: {product_id}) -
{quantity_sold} units sold.")

    print("~" * 30)  # Decorative separator for better readability.

# Function to generate a report of supplier orders.
def generate_supplier_orders_report(orders, suppliers):
    # Print the header for the Supplier Orders Report.
    print("Supplier Orders Report")
    print("~" * 30)  # Decorative separator for better readability.

    supplier_orders = {}  # Dictionary to store the number of orders for
each supplier.

    # Iterate over each order in the list.
    for order in orders:
        supplier_id = order["supplier_id"]  # Get the supplier ID from the
order.

        # Count the number of orders for each supplier.
        if supplier_id in supplier_orders:
            supplier_orders[supplier_id] += 1
        else:
            supplier_orders[supplier_id] = 1

    # Iterate through the supplier_orders dictionary to generate the
report.
    for supplier_id, order_count in supplier_orders.items():
        # Print the supplier name, ID, and total orders placed.
```

```
        print(f"Supplier {suppliers[supplier_id]['name']} (ID:
{supplier_id}) - {order_count} orders placed.")

    print("~" * 30)  # Decorative separator for better readability.


# --- MAIN MENU ---
# Main entry point of the program, provides an interactive menu for the
user to perform various operations.

def main():
    # Initialize the program by loading data for products, suppliers, and
orders.
    products = load_products()  # Load the list of products from storage
(e.g., a file or database).
    suppliers = load_suppliers()  # Load the list of suppliers from
storage.
    orders = load_orders()  # Load the list of orders from storage.

    # Main loop to keep the menu running until the user chooses to exit.
    while True:
        # Display the menu options to the user.
        print("\nMenu:")
        print("1. Add a new product")  # Option to add a new product to the
inventory.
        print("2. Update product details")  # Option to modify existing
product details.
        print("3. Add a new supplier")  # Option to add a new supplier to
the system.
        print("4. Place an order")  # Option to create a new order.
        print("5. Display products inventory")  # Option to view the list
of products.
        print("6. Display suppliers inventory")  # Option to view the list
of suppliers.
        print("7. Generate low stock report")  # Option to generate a
report of products with low stock levels.
        print("8. Generate sales report")  # Option to generate a report on
sales.
        print("9. Generate supplier orders report")  # Option to generate a
report on supplier orders.
        print("10. Exit")  # Option to exit the program.

        # Prompt the user to enter their choice from the menu.
        choice = input("Enter your choice (1-10): ")

        # Handle the user's choice using conditional statements.
        if choice == "1":
```

```
            add_new_product(products)  # Call the function to add a new
product.
        elif choice == "2":
            update_product_details(products)  # Call the function to update
product details.
        elif choice == "3":
            add_new_supplier(suppliers)  # Call the function to add a new
supplier.
        elif choice == "4":
            place_order(products, suppliers, orders)  # Call the function
to place an order.
        elif choice == "5":
            display_products(products)  # Call the function to display the
products inventory.
        elif choice == "6":
            display_suppliers(suppliers)  # Call the function to display
the suppliers inventory.
        elif choice == "7":
            generate_low_stock_report(products)  # Call the function to
generate a low-stock report.
        elif choice == "8":
            generate_sales_report(orders, products)  # Call the function to
generate a sales report.
        elif choice == "9":
            generate_supplier_orders_report(orders, suppliers)  # Call the
function to generate a supplier orders report.
        elif choice == "10":
            print("Exiting. Goodbye!")  # Exit message for the user.
            break  # Break the loop to end the program.
        else:
            print("Invalid choice. Please try again.")  # Error message for
invalid input.

# Run the program
main()  # Start the program by calling the main function.
```

2. products.txt

```
1000|laptop|11500.0|5
1001|mouse|500.0|12
1002|keyboard|494.2|7
1003|web camera|92.0|4
1004|headphones|1200.0|30
```

3.suppliers.txt

```
3000|Selbinyyaz Odekova|01717118385
3001|Dana Matyakubova|01563765652
3002|Wong Li|04653978562
3003|Arazsoltan Oraznepesova|07786543786
3004|Sanjar Annayarov|06764890876
```

4.orders.txt

```
1|1000|2|3000
2|1001|3|3001
3|1002|1|3002
4|1003|1|3003
5|1004|8|3004
```