Lab 6 - Program 2

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

class InsufficientInventoryException extends Exception {
    public InsufficientInventoryException(String message) {
        super(message);
    }
}

class OrderCancellationException extends Exception {
    public OrderCancellationException(String message) {
        super(message);
    }
}

class Item {
    private String name;
    private int quantity;

    public Item(String name, int quantity) {
        this.name = name.toLowerCase(); // Convert to lowercase for case-
insensitive comparison
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public int getQuantity() {
        return quantity;
    }
}

class Order {
    private int orderId;
    private List<Item> items;

    public Order(int orderId, List<Item> items) {
        this.orderId = orderId;
        this.items = items;
    }
    public int getOrderId() {
        return orderId;
    }

    public List<Item> getItems() {
        return items;
    }
}

public class inherit1 {
    private List<Item> inventory = new ArrayList<>();
    private List<Order> orderList = new ArrayList<>();
    private ExecutorService executorService = Executors.newFixedThreadPool(5);
    private static Scanner scanner = new Scanner(System.in);

    public void placeOrder(Order order) {
        orderList.add(order);
        executorService.submit(() -> processOrder(order));
    }

    public void startProcessing() {
        // Start processing orders using worker threads
        executorService.shutdown();
    }

    public void waitForCompletion() {
        try {
            executorService.awaitTermination(Long.MAX_VALUE,
TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public void addInventoryItems() {
        System.out.print("Enter the total number of items to add: ");
        int totalItems = readIntInput();
        scanner.nextLine(); // Consume newline

        for (int i = 0; i < totalItems; i++) {
            System.out.println("Enter details for item " + (i + 1));
            System.out.print("Enter item name: ");
            String itemName = scanner.nextLine();

            System.out.print("Enter quantity: ");
            int quantity = readIntInput();
            //scanner.nextLine(); // Commented out to remove extra newline
            inventory.add(new Item(itemName, quantity));
        }

        System.out.println("Inventory items added successfully.");
    }

    public void updateInventory(Order order) throws
InsufficientInventoryException, OrderCancellationException {
        for (Item orderedItem : order.getItems()) {
            boolean found = false;

            for (Item inventoryItem : inventory) {
                if (inventoryItem.getName().equals(orderedItem.getName())) {
                    found = true;

                    if (inventoryItem.getQuantity() >=
orderedItem.getQuantity()) {
                        inventoryItem = new Item(inventoryItem.getName(),
inventoryItem.getQuantity() - orderedItem.getQuantity());
                    } else {
                        throw new InsufficientInventoryException("Insufficient
inventory for item: " + orderedItem.getName());
                    }
                    break;
                }
            }

            if (!found) {
                throw new OrderCancellationException("Item not found in
inventory: " + orderedItem.getName());
            }
        }
    }

    public boolean checkInventoryAvailability(Item item) {
        for (Item inventoryItem : inventory) {
            if (inventoryItem.getName().equals(item.getName()) &&
inventoryItem.getQuantity() > 0) {
                return true;
            }
        }
        return false;
    }

    public String trackOrderStatus(int orderId) {
        for (Order order : orderList) {
            if (order.getOrderId() == orderId) {
```

```java
                return "Order " + orderId + " is " +
(executorService.isTerminated() ? "processed" : "in progress");
            }
        }
        return "Order " + orderId + " not found";
    }

    private void processOrder(Order order) {
        try {
            updateInventory(order);
            // Simulate order processing time
            TimeUnit.SECONDS.sleep(2);
            System.out.println("Order " + order.getOrderId() + " processed
successfully.");
        } catch (InsufficientInventoryException | OrderCancellationException |
InterruptedException e) {
            System.out.println("Error processing order " + order.getOrderId()
+ ": " + e.getMessage());
        }
    }

    public Order createOrderFromUserInput() {
        System.out.print("Enter order ID: ");
        int orderId = readIntInput();
        scanner.nextLine(); // Consume newline

        System.out.print("Enter the total number of items in the order: ");
        int totalItems = readIntInput();
        scanner.nextLine(); // Consume newline

        List<Item> items = new ArrayList<>();
        for (int i = 0; i < totalItems; i++) {
            System.out.println("Enter details for item " + (i + 1));
            System.out.print("Enter item name: ");
            String itemName = scanner.nextLine();

            System.out.print("Enter quantity: ");
            int quantity = readIntInput();
            //scanner.nextLine(); // Commented out to remove extra newline

            items.add(new Item(itemName, quantity));
        }

        return new Order(orderId, items);
    }

    private static int readIntInput() {
        while (true) {
            try {
                return Integer.parseInt(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid
integer.");
            }
        }
    }

    public void viewOrders() {
        System.out.println("\nList of Orders:");
        for (Order order : orderList) {
            System.out.println("Order ID: " + order.getOrderId());

            System.out.println("Items:");
            for (Item item : order.getItems()) {
                System.out.println("- " + item.getName() + ": " +
item.getQuantity());
            }

            System.out.println("---------------");
        }
    }

    public static void main(String[] args) {
        inherit1 orderSystem = new inherit1();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nSelect an option:");
            System.out.println("1. Add Inventory Items");
            System.out.println("2. View Inventory");
            System.out.println("3. Place Order");
            System.out.println("4. View Orders");
            System.out.println("5. Exit");

            int choice = readIntInput();

            switch (choice) {
                case 1:
                    orderSystem.addInventoryItems();
                    break;
                case 2:
                    orderSystem.displayInventory();
                    break;
                case 3:
                    Order order = orderSystem.createOrderFromUserInput();
                    orderSystem.placeOrder(order);
```

```java
                    break;
                case 4:
                    orderSystem.viewOrders();
                    break;
                case 5:
                    orderSystem.startProcessing();
                    orderSystem.waitForCompletion();
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid
option.");
            }
        }
    }

    private void displayInventory() {
        System.out.println("\nCurrent Inventory:");
        for (Item item : inventory) {
            System.out.println(item.getName() + " - Quantity: " +
item.getQuantity());
        }
    }
}
```

Program 1

```java
package codes;
import java.util.Scanner;
public class lab6a extends Thread
{
    public static void main(String args[])
    {
        lab6a con = new lab6a();
        con.start();
    }

    static int count(int coins[], int n, int sum)
    {

        if (sum == 0) return 1;
        if (sum < 0) return 0;
        if (n <= 0) return 0;
        return count(coins, n - 1, sum)
            + count(coins, n, sum - coins[n - 1]);
    }
    public void run()
    {
        Scanner x = new Scanner(System.in);
        System.out.print("How many types of coins do you have? ");
        int num = x.nextInt();
        int[] coins = new int[num];
        System.out.println("Enter their denominations...");
        for(int i = 0; i<num; i++)
        {
            int incoin;
            while(true)
            {
                incoin = x.nextInt();
                if(incoin<=0)
                    System.out.print("coin denomination cannot be <=0 !!!\n"
                            + "enter again... ");
                else break;
            }
            coins[i] = incoin;
        }
        System.out.print("good. now tell me what sum you want to make... ");
        int sum = x.nextInt();
        x.close();
        int n = coins.length;
        System.out.print("The number of combinations are = "+count(coins, n,
sum));
    }
}
```