//Generate the dataset with random orders for 1000 customers.

//Filter the orders using LINQ to find all orders that are greater than a specified amount (e.g., orders greater than $500).

//Process the dataset concurrently using:

//Parallel programming: Calculate the total amount of all orders using parallel execution.

//Asynchronous programming: Write a method that simulates sending each order to a remote server asynchronously.

//Multithreading: Use multiple threads to display the total number of filtered orders and the total order amount concurrently.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

namespace Prog

{

//create order class

public class Order

{

public int OrderID { get; set; }

public string CustomerName { get; set; }

public double OrderAmount { get; set; }

}

public class Program {

public static async Task Main(string[] args)

{

var orders = GenerateOrders(1000);

var filteredOrders = FilterAmount(orders, 500);

double totalOrderAmount = CalculateTotalAmount(orders);

Console.WriteLine($"Total order amount: {totalOrderAmount}");

DisplayResultsConcurrently(filteredOrders);

}

//generate dataset with random orders for 1000 customers

public static List<Order> GenerateOrders(int numOrders)

{

var random = new Random();

var orders = new List<Order>();

for (int i = 0; i < numOrders; i++)

{

orders.Add(new Order

{

OrderID = i + 1,

CustomerName = "Customer no." + (i + 1),

OrderAmount = random.NextDouble(),

});

}

return orders;

}

//filter orders using LINQ

public static IEnumerable<Order> FilterAmount(List<Order> orders, double amount)

{

return orders.Where(o => o.OrderAmount > amount);

}

//calculate total order with parallel programming

public static double CalculateTotalAmount(List<Order> orders)

{

return orders.AsParallel().Sum(o => o.OrderAmount);

}

//send the orders

public static async Task SendOrder(Order order)

{

Console.WriteLine($"Order {order.OrderID} in queue");

}

//display using multithreading

public static void DisplayFilteredOrders(IEnumerable<Order> filteredOrders)

{

Console.WriteLine($"Total Filtered Orders: {filteredOrders.Count()}");

}

public static void DisplayTotalFilteredOrders(IEnumerable<Order> filteredOrders)

{

Console.WriteLine($"Total Amount of Filtered Orders: {filteredOrders.Sum(o => o.OrderAmount)}");

}

public static void DisplayResultsConcurrently(IEnumerable<Order> filteredOrders)

{

Thread countThread = new Thread(() => DisplayFilteredOrders(filteredOrders));

Thread amountThread = new Thread(() => DisplayTotalFilteredOrders(filteredOrders));

countThread.Start();

amountThread.Start();

countThread.Join();

amountThread.Join();

}

}

}