using MultithreadingApp;

using System.Diagnostics.Metrics;

using System.Diagnostics.SymbolStore;

public class Program

{

public static void Main()

{

//Console.WriteLine("Hello, World!");

//var bgThread = new Thread((object? data) =>

//{

// if (data is null) return;

// int counter = 0;

// var result = int.TryParse(data.ToString(), out int maxCount);

// if (!result) return;

// while (counter < maxCount)

// {

// bool isNetwork = System.Net.NetworkInformation

// .NetworkInterface.GetIsNetworkAvailable();

// Console.WriteLine($"Is Network Available? Answer: {isNetwork}");

// Thread.Sleep(100);

// counter++;

// }

//});

//bgThread.IsBackground= true;

//bgThread.Start(12);

//for (int i = 0; i < 10; i++)

//{

// Console.WriteLine($"Main Thread is running {i}");

// Thread.Sleep(100);

//}

//Console.WriteLine("Done");

//Console.ReadKey();

var network = new NetworkStatusChecker();

var bgThread1 = new Thread(network.CheckNetworkStatus);

var bgThread2 = new Thread(network.CheckNetworkStatus);

var bgThread3 = new Thread(network.CheckNetworkStatus);

var bgThread4 = new Thread(network.CheckNetworkStatus);

var bgThread5 = new Thread(network.CheckNetworkStatus);

bgThread1.Priority = ThreadPriority.Lowest;

bgThread2.Priority = ThreadPriority.BelowNormal;

bgThread3.Priority = ThreadPriority.Normal;

bgThread4.Priority = ThreadPriority.AboveNormal;

bgThread5.Priority = ThreadPriority.Highest;

bgThread1.Start("Lowest");

bgThread2.Start("Below");

bgThread3.Start("Normal");

bgThread4.Start("Above");

bgThread5.Start("High");

for (int i = 0; i < 10; i++)

{

Console.WriteLine($"Main Thread is running {i}");

}

Console.WriteLine("Done");

Console.ReadKey();

}

}

// ThreadState Enum Values:

// Aborted

// AbortRequested

// Background

// Running

// Stopped

// StopRequested

// Suspended

// SuspendRequested

// Unstarted

// WaitSleepJoin: the thread is currently blocked

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MultithreadingApp

{

public class NetworkStatusChecker

{

public void CheckNetworkStatus(object data)

{

for (int i = 0; i < 12; i++)

{

bool isNetwork = System.Net.NetworkInformation

.NetworkInterface.GetIsNetworkAvailable();

Console.WriteLine($"Is Network Available? " +

$"Thread {(string) data} Answer: {isNetwork}");

Thread.Sleep(100);

}

}

public void ChechNetworkStatus2(object data)

{

bool finish = false;

var canceltoken = (CancellationToken)data;

canceltoken.Register(() => { finish = true; });

while (!finish)

{

bool isNetwork = System.Net.NetworkInformation

.NetworkInterface.GetIsNetworkAvailable();

Console.WriteLine($"Is Network Available? " +

$"Thread {(string)data} Answer: {isNetwork}");

}

}

}

}

var iList = new List<int>() { 1, 2, 3, 4};

var firstIndex = iList[0]; // immediate execution

iList.Remove(0);

firstIndex.Dump(); // deferred execution: firstIndex assignment will NOT run until called.

IQueryable result = Employees.Select(x => x);

var getT = result.GetType();

getT.Dump();

result.Dump(); // it ran the assignment as if it was assigned at this location

// Equivalent to result = Employees.Select(x => x); result.Dump;

foreach (var item in Employees.Select(x => x))

{

if (item.EmployeeID == 1) item.FirstName = "Pineapple";

}

result.Dump(); // it ran the assignment as if it was assigned at this location

// Immediate Execution

var anotherQuery = Products.Select(x => x).ToList();

var getTOfAnother = anotherQuery.GetType();

anotherQuery.Dump();

getTOfAnother.Dump();

foreach (var item in anotherQuery)

{

if (item.ProductID == 1) item.ProductName = "Pineapple";

}

anotherQuery.Dump();

var NewRes = Products.Select(x => x);

NewRes.Dump;

// Deferred: Select, Where, Take, Skip... Querying

// Immediate: Aggregates and Conversion of IQueryable to another from such as ToList, ToArray