| NAME | RAJ SANJAY JADHAV |
| --- | --- |
| CLASS | BE-3-CSE |
| BATCH | B |
| ROLL NO | 512024 |
| PRN | 2019033800129402 |

**PRACTICAL-6(A)**

**GIT REPOSITORY :** [**PRACTICAL-6**](https://github.com/Rajjadhav1710/.NET_Practical/tree/main/Practical-6)

**COLLECTIONS**

**BIT ARRAY SAMPLE**

**CODE:**

**using System;**

**using System.Collections;**

**using System.Collections.Specialized;**

**using System.Text;**

**namespace BitArraySample**

**{**

**class Program**

**{**

**static void BitArrayDemo()**

**{**

**var bits1 = new BitArray(8);**

**bits1.SetAll(true);**

**bits1.Set(1, false);**

**bits1[5] = false;**

**bits1[7] = false;**

**Console.Write("initialized: ");**

**DisplayBits(bits1);**

**Console.WriteLine();**

**DisplayBits(bits1);**

**bits1.Not();**

**Console.Write(" not ");**

**DisplayBits(bits1);**

**Console.WriteLine();**

**var bits2 = new BitArray(bits1);**

**bits2[0] = true;**

**bits2[1] = false;**

**bits2[4] = true;**

**DisplayBits(bits1);**

**Console.Write(" or ");**

**DisplayBits(bits2);**

**Console.Write(" : ");**

**bits1.Or(bits2);**

**DisplayBits(bits1);**

**Console.WriteLine();**

**DisplayBits(bits2);**

**Console.Write(" and ");**

**DisplayBits(bits1);**

**Console.Write(" : ");**

**bits2.And(bits1);**

**DisplayBits(bits2);**

**Console.WriteLine();**

**DisplayBits(bits1);**

**Console.Write(" xor ");**

**DisplayBits(bits2);**

**bits1.Xor(bits2);**

**Console.Write(" : ");**

**DisplayBits(bits1);**

**Console.WriteLine();**

**}**

**static void BitVectorDemo()**

**{**

**var bits1 = new BitVector32();**

**int bit1 = BitVector32.CreateMask();**

**int bit2 = BitVector32.CreateMask(bit1);**

**int bit3 = BitVector32.CreateMask(bit2);**

**int bit4 = BitVector32.CreateMask(bit3);**

**int bit5 = BitVector32.CreateMask(bit4);**

**bits1[bit1] = true;**

**bits1[bit2] = false;**

**bits1[bit3] = true;**

**bits1[bit4] = true;**

**Console.WriteLine(bits1);**

**bits1[0xabcdef] = true;**

**Console.WriteLine(bits1);**

**int received = 0x79abcdef;**

**var bits2 = new BitVector32(received);**

**Console.WriteLine(bits2);**

**// sections: FF EEE DDD CCCC BBBBBBBB AAAAAAAAAAAA**

**BitVector32.Section sectionA = BitVector32.CreateSection(0xfff);**

**BitVector32.Section sectionB = BitVector32.CreateSection(0xff, sectionA);**

**BitVector32.Section sectionC = BitVector32.CreateSection(0xf, sectionB);**

**BitVector32.Section sectionD = BitVector32.CreateSection(0x7, sectionC);**

**BitVector32.Section sectionE = BitVector32.CreateSection(0x7, sectionD);**

**BitVector32.Section sectionF = BitVector32.CreateSection(0x3, sectionE);**

**Console.WriteLine("Section A: " + IntToBinaryString(bits2[sectionA], true));**

**Console.WriteLine("Section B: " + IntToBinaryString(bits2[sectionB], true));**

**Console.WriteLine("Section C: " + IntToBinaryString(bits2[sectionC], true));**

**Console.WriteLine("Section D: " + IntToBinaryString(bits2[sectionD], true));**

**Console.WriteLine("Section E: " + IntToBinaryString(bits2[sectionE], true));**

**Console.WriteLine("Section F: " + IntToBinaryString(bits2[sectionF], true));**

**}**

**static string IntToBinaryString(int bits, bool removeTrailingZero)**

**{**

**var sb = new StringBuilder(32);**

**for (int i = 0; i < 32; i++)**

**{**

**if ((bits & 0x80000000) != 0)**

**{**

**sb.Append("1");**

**}**

**else**

**{**

**sb.Append("0");**

**}**

**bits = bits << 1;**

**}**

**string s = sb.ToString();**

**if (removeTrailingZero)**

**return s.TrimStart('0');**

**else**

**return s;**

**}**

**static void Main()**

**{**

**// BitArrayDemo();**

**BitVectorDemo();**

**}**

**static void DisplayBits(BitArray bits)**

**{**

**foreach (bool bit in bits)**

**{**

**Console.Write(bit ? 1 : 0);**

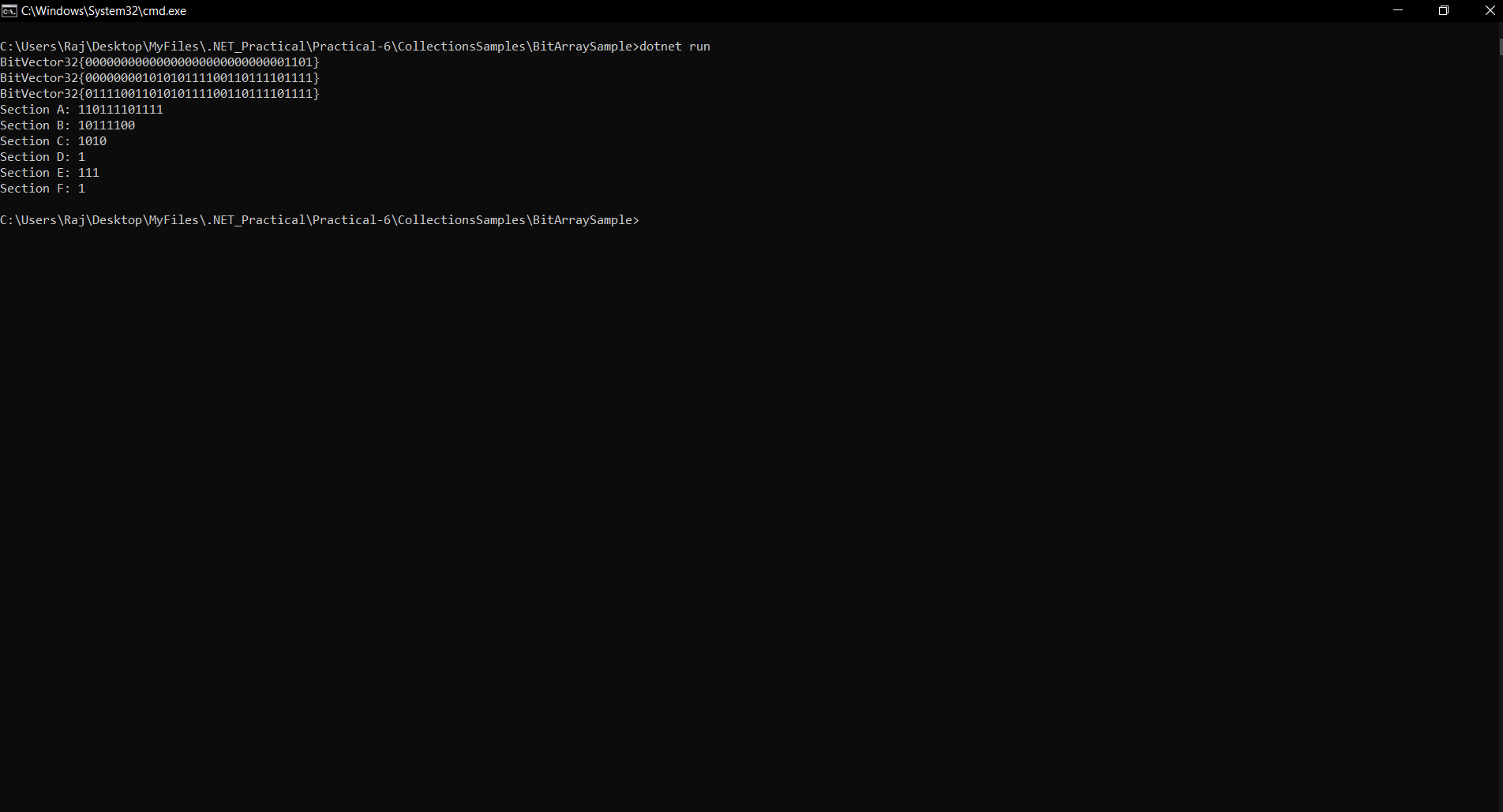
**}**

**}**

**}**

**}**

**OUTPUT:**

****

**CONCURRENT SAMPLE**

**CODE:**

**using System;**

**using System.Collections.Concurrent;**

**using System.Threading;**

**namespace ConcurrentSample**

**{**

**class Program**

**{**

**static void Main()**

**{**

**// BlockingDemo();**

**BlockingDemoSimple();**

**}**

**static void BlockingDemoSimple()**

**{**

**var sharedCollection = new BlockingCollection<int>();**

**var events = new ManualResetEventSlim[2];**

**var waits = new WaitHandle[2];**

**for (int i = 0; i < 2; i++)**

**{**

**events[i] = new ManualResetEventSlim(false);**

**waits[i] = events[i].WaitHandle;**

**}**

**var producer = new Thread(obj =>**

**{**

**var state = (Tuple<BlockingCollection<int>, ManualResetEventSlim>)obj;**

**var coll = state.Item1;**

**var ev = state.Item2;**

**var r = new Random();**

**for (int i = 0; i < 300; i++)**

**{**

**coll.Add(r.Next(3000));**

**}**

**ev.Set();**

**});**

**producer.Start(Tuple.Create<BlockingCollection<int>, ManualResetEventSlim>(sharedCollection, events[0]));**

**var consumer = new Thread(obj =>**

**{**

**var state = (Tuple<BlockingCollection<int>, ManualResetEventSlim>)obj;**

**var coll = state.Item1;**

**var ev = state.Item2;**

**for (int i = 0; i < 300; i++)**

**{**

**int result = coll.Take();**

**}**

**ev.Set();**

**});**

**consumer.Start(Tuple.Create<BlockingCollection<int>, ManualResetEventSlim>(sharedCollection, events[1]));**

**if (!WaitHandle.WaitAll(waits))**

**Console.WriteLine("wait failed");**

**else**

**Console.WriteLine("reading/writing finished");**

**}**

**static void BlockingDemo()**

**{**

**const int threadCount = 10;**

**ManualResetEventSlim[] events = new ManualResetEventSlim[threadCount];**

**WaitHandle[] waits = new WaitHandle[threadCount];**

**var consoleLock = new object();**

**for (int thread = 0; thread < threadCount; thread++)**

**{**

**events[thread] = new ManualResetEventSlim(false);**

**waits[thread] = events[thread].WaitHandle;**

**}**

**var sharedCollection = new BlockingCollection<int>();**

**for (int thread = 0; thread < threadCount >> 1; thread++)**

**{**

**var producer = new Thread((state) =>**

**{**

**var coll = ((Tuple<BlockingCollection<int>, ManualResetEventSlim>)state).Item1;**

**var wait = ((Tuple<BlockingCollection<int>, ManualResetEventSlim>)state).Item2;**

**var r = new Random();**

**for (int i = 0; i < 300; i++)**

**{**

**int data = r.Next(30000);**

**if (!coll.TryAdd(data))**

**{**

**Console.WriteLine("\*\*\*\* couldn't add");**

**}**

**else**

**{**

**lock (consoleLock)**

**{**

**Console.ForegroundColor = ConsoleColor.Cyan;**

**Console.Write(" {0} ", data);**

**Console.ResetColor();**

**}**

**}**

**Thread.Sleep(r.Next(40));**

**}**

**wait.Set();**

**});**

**producer.Start(Tuple.Create<BlockingCollection<int>, ManualResetEventSlim>(sharedCollection, events[thread]));**

**}**

**Thread.Sleep(500); // give the producers a headstart**

**for (int thread = threadCount >> 1; thread < threadCount; thread++)**

**{**

**var consumer = new Thread((state) =>**

**{**

**var coll = ((Tuple<BlockingCollection<int>, ManualResetEventSlim>)state).Item1;**

**var wait = ((Tuple<BlockingCollection<int>, ManualResetEventSlim>)state).Item2;**

**var r = new Random();**

**for (int i = 0; i < 3000; i++)**

**{**

**int result;**

**if (!coll.TryTake(out result))**

**{**

**Console.WriteLine("couldn't take");**

**}**

**else**

**{**

**lock (consoleLock)**

**{**

**Console.ForegroundColor = ConsoleColor.Red;**

**Console.WriteLine(" {0} ", result);**

**Console.ResetColor();**

**}**

**}**

**Thread.Sleep(r.Next(40));**

**}**

**wait.Set();**

**});**

**consumer.Start(Tuple.Create<BlockingCollection<int>, ManualResetEventSlim>(sharedCollection, events[thread]));**

**}**

**if (!WaitHandle.WaitAll(waits))**

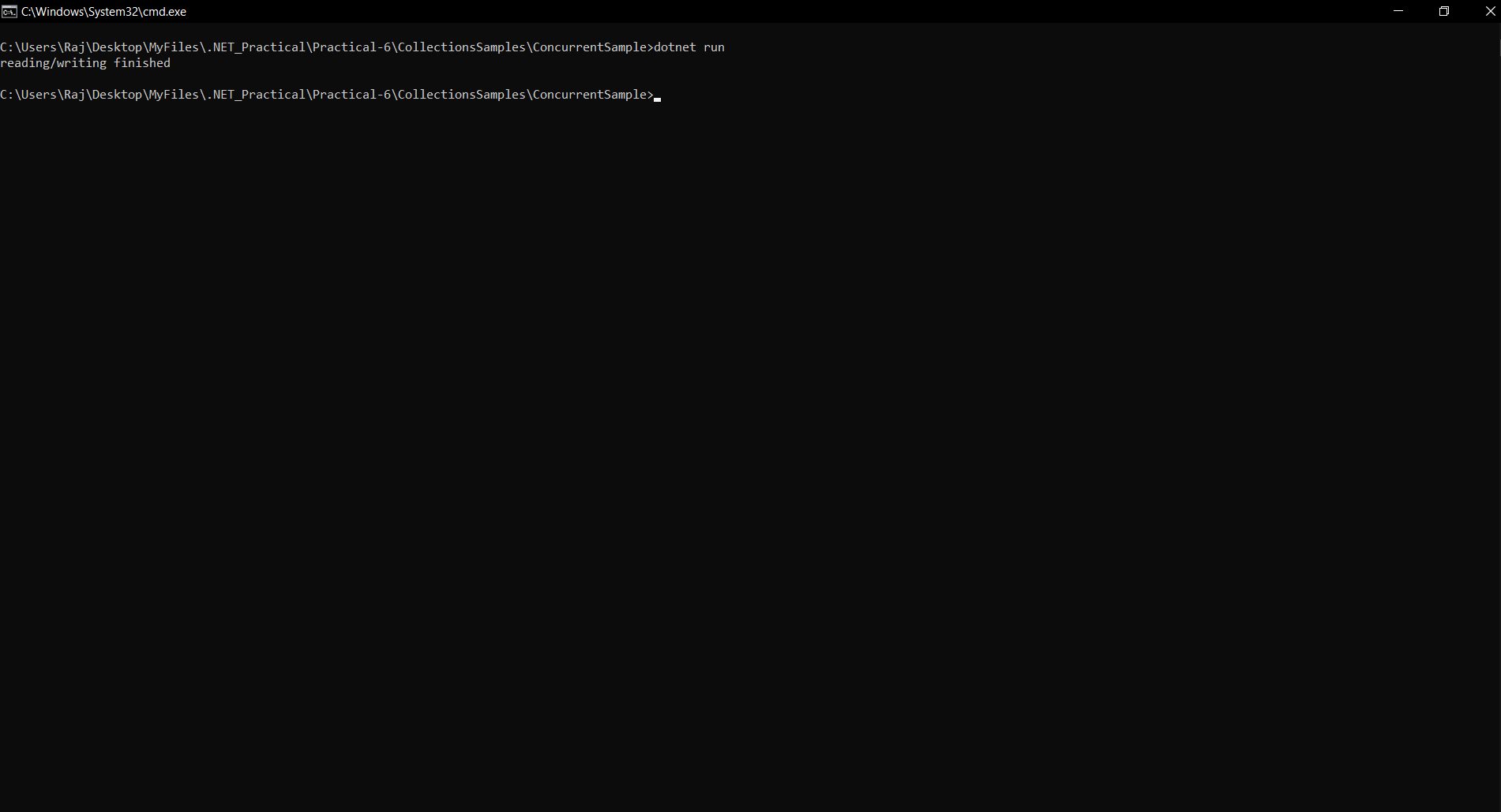
**Console.WriteLine("error waiting...");**

**}**

**}**

**}**

**OUTPUT:**

****

**DICTIONARY SAMPLE**

**CODE:**

**Employee.cs**

**using System;**

**namespace Wrox.ProCSharp.Collections**

**{**

**[Serializable]**

**public class Employee**

**{**

**private string name;**

**private decimal salary;**

**private readonly EmployeeId id;**

**public Employee(EmployeeId id, string name, decimal salary)**

**{**

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public override string ToString()**

**{**

**return String.Format("{0}: {1, -20} {2:C}",**

**id.ToString(), name, salary);**

**}**

**}**

**}**

**EmployeeId.cs**

**using System;**

**namespace Wrox.ProCSharp.Collections**

**{**

**[Serializable]**

**public class EmployeeIdException : Exception**

**{**

**public EmployeeIdException(string message) : base(message) { }**

**}**

**[Serializable]**

**public struct EmployeeId : IEquatable<EmployeeId>**

**{**

**private readonly char prefix;**

**private readonly int number;**

**public EmployeeId(string id)**

**{**

**if (id == null) throw new ArgumentNullException("id");**

**prefix = (id.ToUpper())[0];**

**int numLength = id.Length - 1;**

**try**

**{**

**number = int.Parse(id.Substring(1, numLength > 6 ? 6 : numLength));**

**}**

**catch (FormatException)**

**{**

**throw new EmployeeIdException("Invalid EmployeeId format");**

**}**

**}**

**public override string ToString()**

**{**

**return prefix.ToString() + string.Format("{0,6:000000}", number);**

**}**

**public override int GetHashCode()**

**{**

**return (number ^ number << 16) \* 0x15051505;**

**}**

**public bool Equals(EmployeeId other)**

**{**

**if (other == null) return false;**

**return (prefix == other.prefix && number == other.number);**

**}**

**public override bool Equals(object obj)**

**{**

**return Equals((EmployeeId)obj);**

**}**

**public static bool operator ==(EmployeeId left, EmployeeId right)**

**{**

**return left.Equals(right);**

**}**

**public static bool operator !=(EmployeeId left, EmployeeId right)**

**{**

**return !(left == right);**

**}**

**}**

**}**

**Program.cs**

**using System;**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var employees = new Dictionary<EmployeeId, Employee>(31);**

**var idKyle = new EmployeeId("T3755");**

**var kyle = new Employee(idKyle, "Kyle Bush", 5443890.00m);**

**employees.Add(idKyle, kyle);**

**Console.WriteLine(kyle);**

**var idCarl = new EmployeeId("F3547");**

**var carl = new Employee(idCarl, "Carl Edwards", 5597120.00m);**

**employees.Add(idCarl, carl);**

**Console.WriteLine(carl);**

**var idJimmie = new EmployeeId("C3386");**

**var jimmie = new Employee(idJimmie, "Jimmie Johnson", 5024710.00m);**

**employees.Add(idJimmie, jimmie);**

**Console.WriteLine(jimmie);**

**var idDale = new EmployeeId("C3323");**

**var dale = new Employee(idDale, "Dale Earnhardt Jr.", 3522740.00m);**

**employees[idDale] = dale;**

**Console.WriteLine(dale);**

**var idJeff = new EmployeeId("C3234");**

**var jeff = new Employee(idJeff, "Jeff Burton", 3879540.00m);**

**employees[idJeff] = jeff;**

**Console.WriteLine(jeff);**

**while (true)**

**{**

**Console.Write("Enter employee id (X to exit)> ");**

**var userInput = Console.ReadLine();**

**userInput = userInput.ToUpper();**

**if (userInput == "X") break;**

**EmployeeId id;**

**try**

**{**

**id = new EmployeeId(userInput);**

**Employee employee;**

**if (!employees.TryGetValue(id, out employee))**

**{**

**Console.WriteLine("Employee with id {0} does not exist", id);**

**}**

**else**

**{**

**Console.WriteLine(employee);**

**}**

**}**

**catch (EmployeeIdException ex)**

**{**

**Console.WriteLine(ex.Message);**

**}**

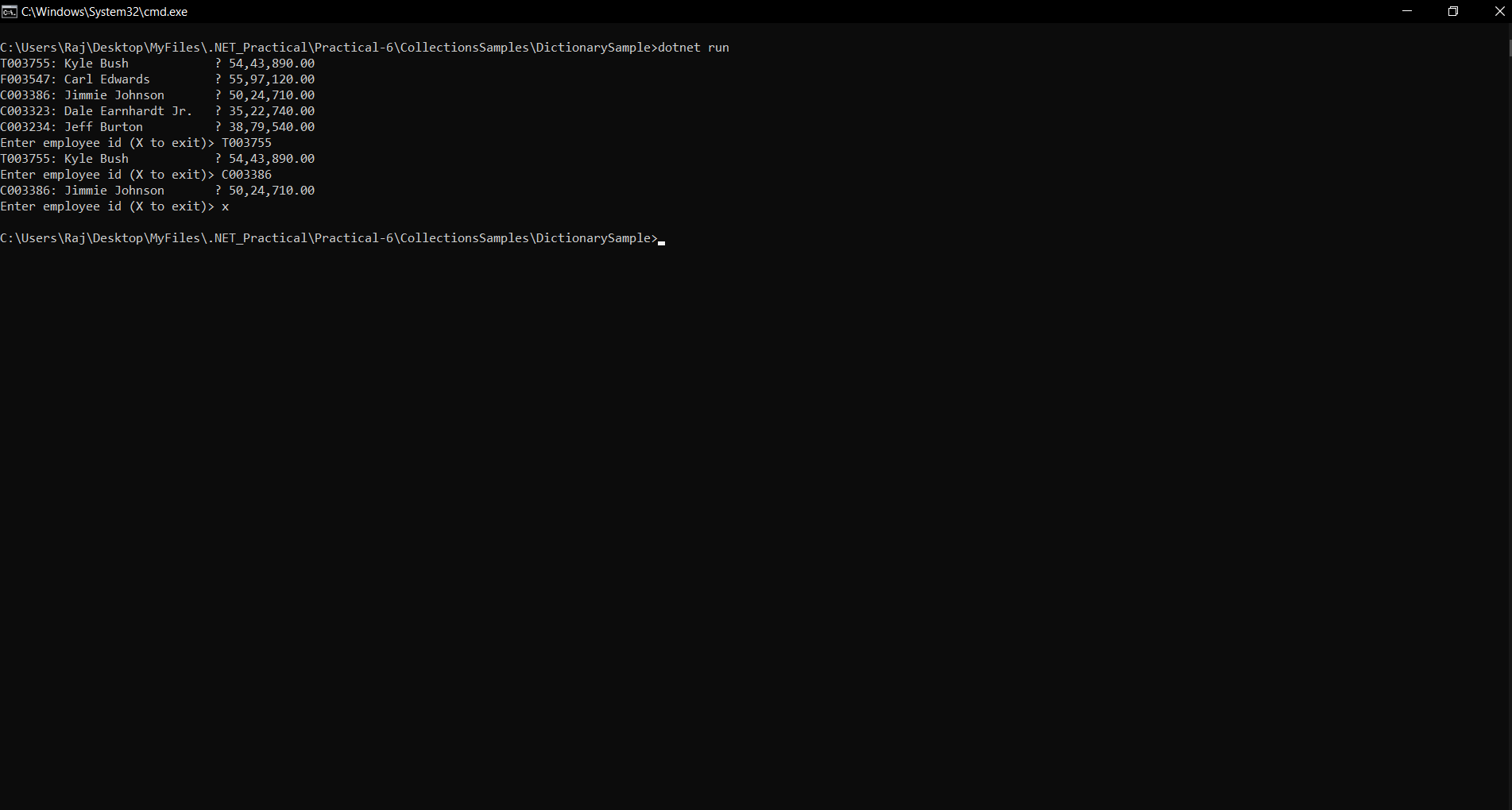
**}**

**}**

**}**

**}**

**OUTPUT:**

****

**LINKED LIST SAMPLE**

**CODE:**

**Document.cs**

**namespace Wrox.ProCSharp.Collections**

**{**

**public class Document**

**{**

**public string Title { get; private set; }**

**public string Content { get; private set; }**

**public byte Priority { get; private set; }**

**public Document(string title, string content, byte priority = 0)**

**{**

**this.Title = title;**

**this.Content = content;**

**this.Priority = priority;**

**}**

**}**

**}**

**PriorityDocumentManager.cs**

**using System;**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**public class PriorityDocumentManager**

**{**

**private readonly LinkedList<Document> documentList;**

**// priorities 0.9**

**private readonly List<LinkedListNode<Document>> priorityNodes;**

**public PriorityDocumentManager()**

**{**

**documentList = new LinkedList<Document>();**

**priorityNodes = new List<LinkedListNode<Document>>(10);**

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

**{**

**priorityNodes.Add(new LinkedListNode<Document>(null));**

**}**

**}**

**public void AddDocument(Document d)**

**{**

**if (d == null) throw new ArgumentNullException("d");**

**AddDocumentToPriorityNode(d, d.Priority);**

**}**

**private void AddDocumentToPriorityNode(Document doc, int priority)**

**{**

**if (priority > 9 || priority < 0)**

**throw new ArgumentException("Priority must be between 0 and 9");**

**if (priorityNodes[priority].Value == null)**

**{**

**--priority;**

**if (priority >= 0)**

**{**

**// check for the next lower priority**

**AddDocumentToPriorityNode(doc, priority);**

**}**

**else // now no priority node exists with the same priority or lower**

**// add the new document to the end**

**{**

**documentList.AddLast(doc);**

**priorityNodes[doc.Priority] = documentList.Last;**

**}**

**return;**

**}**

**else // a priority node exists**

**{**

**LinkedListNode<Document> prioNode = priorityNodes[priority];**

**if (priority == doc.Priority)**

**// priority node with the same priority exists**

**{**

**documentList.AddAfter(prioNode, doc);**

**// set the priority node to the last document with the same priority**

**priorityNodes[doc.Priority] = prioNode.Next;**

**}**

**else // only priority node with a lower priority exists**

**{**

**// get the first node of the lower priority**

**LinkedListNode<Document> firstPrioNode = prioNode;**

**while (firstPrioNode.Previous != null &&**

**firstPrioNode.Previous.Value.Priority == prioNode.Value.Priority)**

**{**

**firstPrioNode = prioNode.Previous;**

**prioNode = firstPrioNode;**

**}**

**documentList.AddBefore(firstPrioNode, doc);**

**// set the priority node to the new value**

**priorityNodes[doc.Priority] = firstPrioNode.Previous;**

**}**

**}**

**}**

**public void DisplayAllNodes()**

**{**

**foreach (Document doc in documentList)**

**{**

**Console.WriteLine("priority: {0}, title {1}", doc.Priority, doc.Title);**

**}**

**}**

**// returns the document with the highest priority**

**// (that's first in the linked list)**

**public Document GetDocument()**

**{**

**Document doc = documentList.First.Value;**

**documentList.RemoveFirst();**

**return doc;**

**}**

**}**

**}**

**Program.cs**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**PriorityDocumentManager pdm = new PriorityDocumentManager();**

**pdm.AddDocument(new Document("one", "Sample", 8));**

**pdm.AddDocument(new Document("two", "Sample", 3));**

**pdm.AddDocument(new Document("three", "Sample", 4));**

**pdm.AddDocument(new Document("four", "Sample", 8));**

**pdm.AddDocument(new Document("five", "Sample", 1));**

**pdm.AddDocument(new Document("six", "Sample", 9));**

**pdm.AddDocument(new Document("seven", "Sample", 1));**

**pdm.AddDocument(new Document("eight", "Sample", 1));**

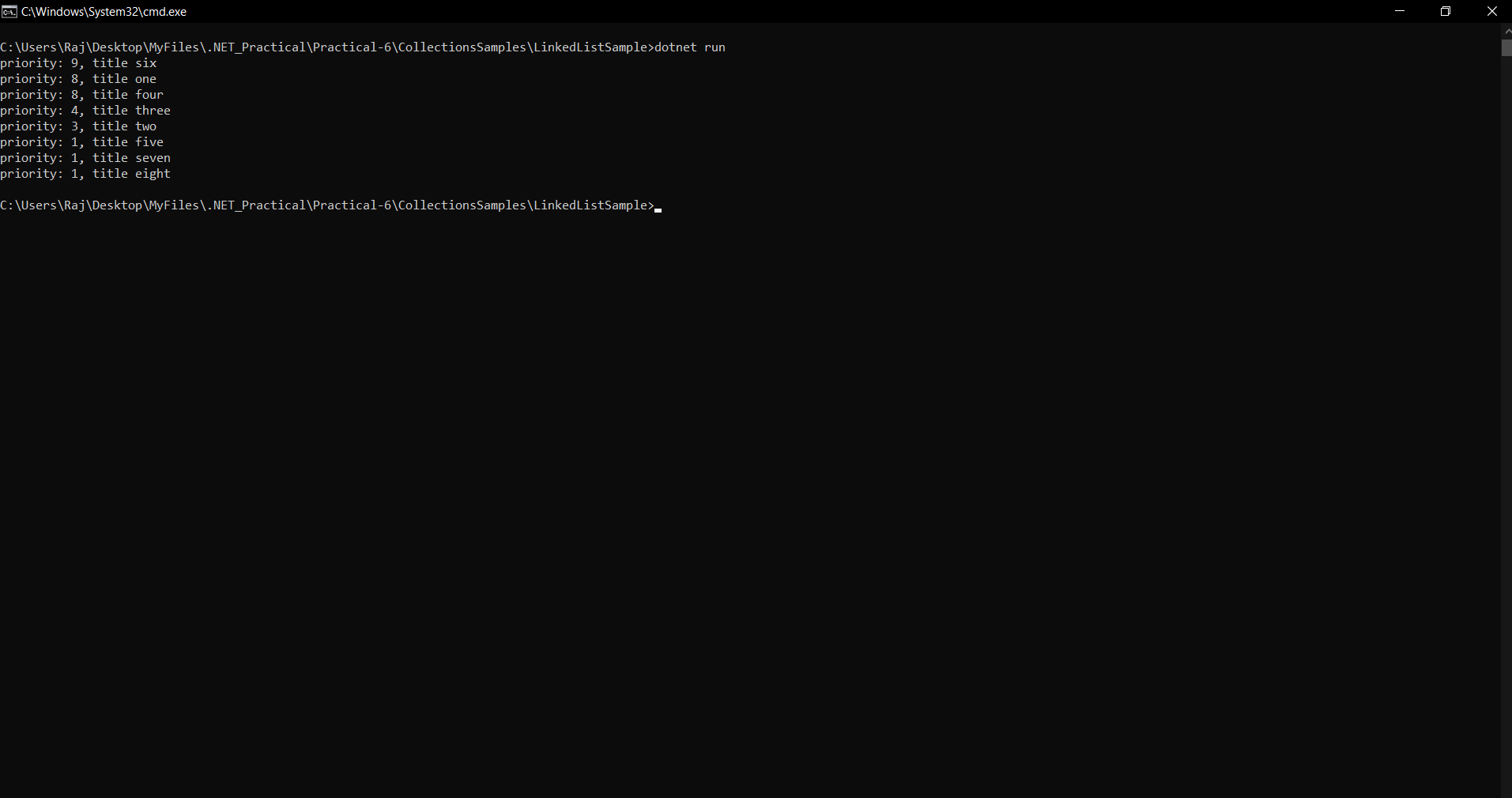
**pdm.DisplayAllNodes();**

**}**

**}**

**}**

**OUTPUT:**

****

**LIST SAMPLE**

**CODE:**

**Program.cs**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var graham = new Racer(7, "Graham", "Hill", "UK", 14);**

**var emerson = new Racer(13, "Emerson", "Fittipaldi", "Brazil", 14);**

**var mario = new Racer(16, "Mario", "Andretti", "USA", 12);**

**var racers = new List<Racer>(20) { graham, emerson, mario };**

**racers.Add(new Racer(24, "Michael", "Schumacher", "Germany", 91));**

**racers.Add(new Racer(27, "Mika", "Hakkinen", "Finland", 20));**

**racers.AddRange(new Racer[] {**

**new Racer(14, "Niki", "Lauda", "Austria", 25),**

**new Racer(21, "Alain", "Prost", "France", 51)});**

**var racers2 = new List<Racer>(new Racer[] {**

**new Racer(12, "Jochen", "Rindt", "Austria", 6),**

**new Racer(22, "Ayrton", "Senna", "Brazil", 41) });**

**}**

**}**

**}**

**Racer.cs**

**using System;**

**namespace Wrox.ProCSharp.Collections**

**{**

**[Serializable]**

**public class Racer : IComparable<Racer>, IFormattable**

**{**

**public int Id { get; private set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string Country { get; set; }**

**public int Wins { get; set; }**

**public Racer(int id, string firstName, string lastName, string country = null, int wins = 0)**

**{**

**this.Id = id;**

**this.FirstName = firstName;**

**this.LastName = lastName;**

**this.Country = country;**

**this.Wins = wins;**

**}**

**public override string ToString()**

**{**

**return String.Format("{0} {1}", FirstName, LastName);**

**}**

**public string ToString(string format, IFormatProvider formatProvider)**

**{**

**switch (format.ToUpper())**

**{**

**case null:**

**case "N": // name**

**return ToString();**

**case "F": // first name**

**return FirstName;**

**case "L": // last name**

**return LastName;**

**case "W": // Wins**

**return String.Format("{0}, Wins: {1}", ToString(), Wins);**

**case "C": // Country**

**return String.Format("{0}, Country: {1}", ToString(), Country);**

**case "A": // All**

**return String.Format("{0}, {1} Wins: {2}", ToString(), Country, Wins);**

**default:**

**throw new FormatException(String.Format(formatProvider,**

**"Format {0} is not supported", format));**

**}**

**}**

**public string ToString(string format)**

**{**

**return ToString(format, null);**

**}**

**public int CompareTo(Racer other)**

**{**

**int compare = this.LastName.CompareTo(other.LastName);**

**if (compare == 0)**

**return this.FirstName.CompareTo(other.FirstName);**

**return compare;**

**}**

**}**

**}**

**RacerComparer.cs**

**using System;**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**public enum CompareType**

**{**

**FirstName,**

**LastName,**

**Country,**

**Wins**

**}**

**public class RacerComparer : IComparer<Racer>**

**{**

**private CompareType compareType;**

**public RacerComparer(CompareType compareType)**

**{**

**this.compareType = compareType;**

**}**

**public int Compare(Racer x, Racer y)**

**{**

**if (x == null) throw new ArgumentNullException("x");**

**if (y == null) throw new ArgumentNullException("y");**

**int result;**

**switch (compareType)**

**{**

**case CompareType.FirstName:**

**return x.FirstName.CompareTo(y.FirstName);**

**case CompareType.LastName:**

**return x.LastName.CompareTo(y.LastName);**

**case CompareType.Country:**

**if ((result = x.Country.CompareTo(y.Country)) == 0)**

**return x.LastName.CompareTo(y.LastName);**

**else**

**return result;**

**case CompareType.Wins:**

**return x.Wins.CompareTo(y.Wins);**

**default:**

**throw new ArgumentException("Invalid Compare Type");**

**}**

**}**

**}**

**}**

**LOOKUP SAMPLE**

**CODE:**

**Program.cs**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var racers = new List<Racer>();**

**racers.Add(new Racer(26, "Jacques", "Villeneuve", "Canada", 11));**

**racers.Add(new Racer(18, "Alan", "Jones", "Australia", 12));**

**racers.Add(new Racer(11, "Jackie", "Stewart", "United Kingdom", 27));**

**racers.Add(new Racer(15, "James", "Hunt", "United Kingdom", 10));**

**racers.Add(new Racer(5, "Jack", "Brabham", "Australia", 14));**

**var lookupRacers = racers.ToLookup(r => r.Country);**

**foreach (Racer r in lookupRacers["Australia"])**

**{**

**Console.WriteLine(r);**

**}**

**}**

**}**

**}**

**Racer.cs**

**using System;**

**namespace Wrox.ProCSharp.Collections**

**{**

**[Serializable]**

**public class Racer : IComparable<Racer>, IFormattable**

**{**

**public int Id { get; private set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string Country { get; set; }**

**public int Wins { get; set; }**

**public Racer(int id, string firstName, string lastName, string country = null, int wins = 0)**

**{**

**this.Id = id;**

**this.FirstName = firstName;**

**this.LastName = lastName;**

**this.Country = country;**

**this.Wins = wins;**

**}**

**public override string ToString()**

**{**

**return String.Format("{0} {1}", FirstName, LastName);**

**}**

**public string ToString(string format, IFormatProvider formatProvider)**

**{**

**if (format == null) format = "N";**

**switch (format.ToUpper())**

**{**

**case "N": // name**

**return ToString();**

**case "F": // first name**

**return FirstName;**

**case "L": // last name**

**return LastName;**

**case "W": // Wins**

**return String.Format("{0}, Wins: {1}", ToString(), Wins);**

**case "C": // Country**

**return String.Format("{0}, Country: {1}", ToString(), Country);**

**case "A": // All**

**return String.Format("{0}, {1} Wins: {2}", ToString(), Country, Wins);**

**default:**

**throw new FormatException(String.Format(formatProvider,**

**"Format {0} is not supported", format));**

**}**

**}**

**public string ToString(string format)**

**{**

**return ToString(format, null);**

**}**

**public int CompareTo(Racer other)**

**{**

**int compare = this.LastName.CompareTo(other.LastName);**

**if (compare == 0)**

**return this.FirstName.CompareTo(other.FirstName);**

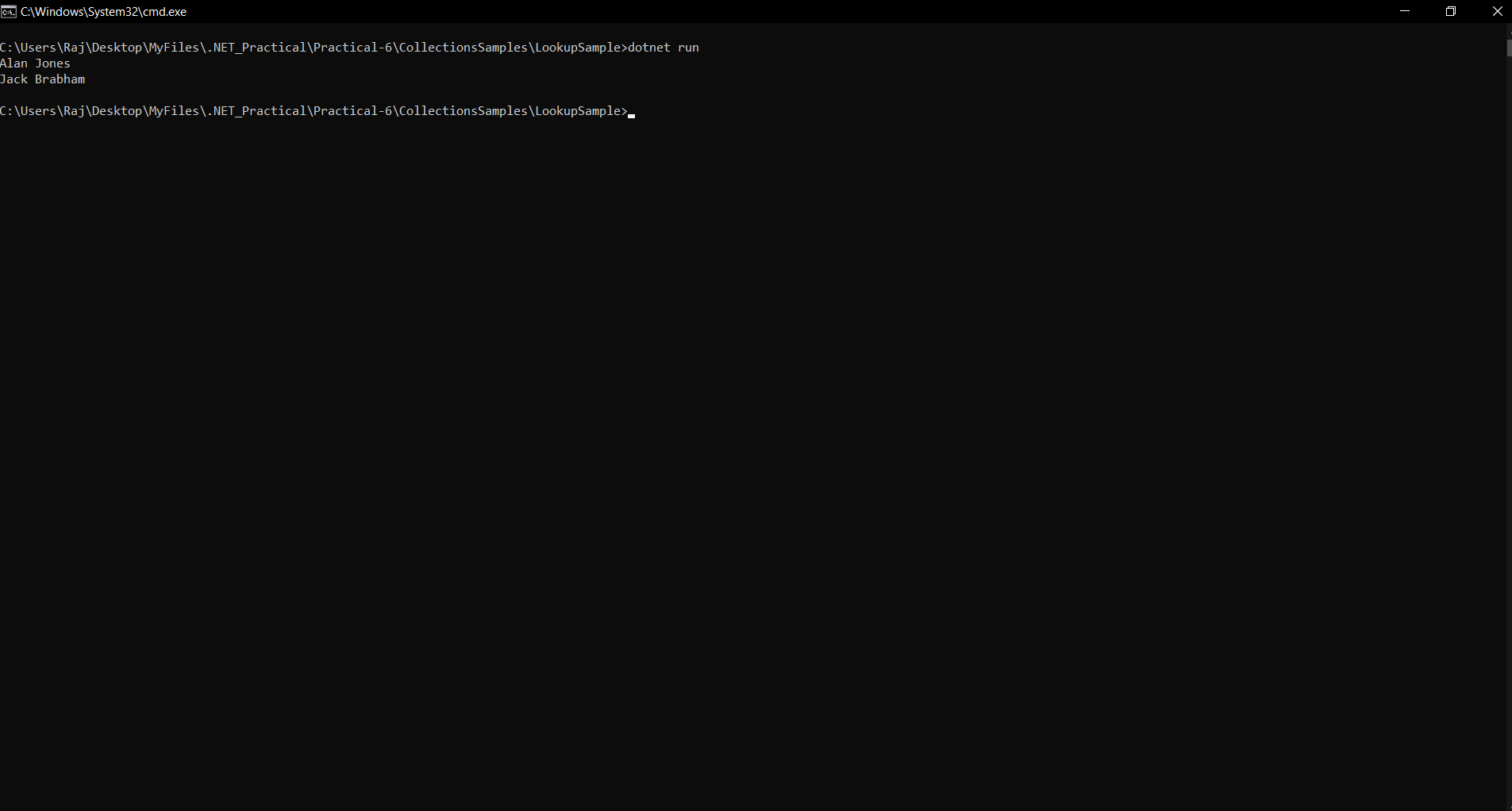
**return compare;**

**}**

**}**

**}**

**OUTPUT:**

****

**OBSERVABLE COLLECTION SAMPLE**

**CODE:**

**using System;**

**using System.Collections.ObjectModel;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var data = new ObservableCollection<string>();**

**data.CollectionChanged += Data\_CollectionChanged;**

**data.Add("One");**

**data.Add("Two");**

**data.Insert(1, "Three");**

**data.Remove("One");**

**}**

**static void Data\_CollectionChanged(object sender, System.Collections.Specialized.NotifyCollectionChangedEventArgs e)**

**{**

**Console.WriteLine("action: {0}", e.Action.ToString());**

**if (e.OldItems != null)**

**{**

**Console.WriteLine("starting index for old item(s): {0}", e.OldStartingIndex);**

**Console.WriteLine("old item(s):");**

**foreach (var item in e.OldItems)**

**{**

**Console.WriteLine(item);**

**}**

**}**

**if (e.NewItems != null)**

**{**

**Console.WriteLine("starting index for new item(s): {0}", e.NewStartingIndex);**

**Console.WriteLine("new item(s): ");**

**foreach (var item in e.NewItems)**

**{**

**Console.WriteLine(item);**

**}**

**}**

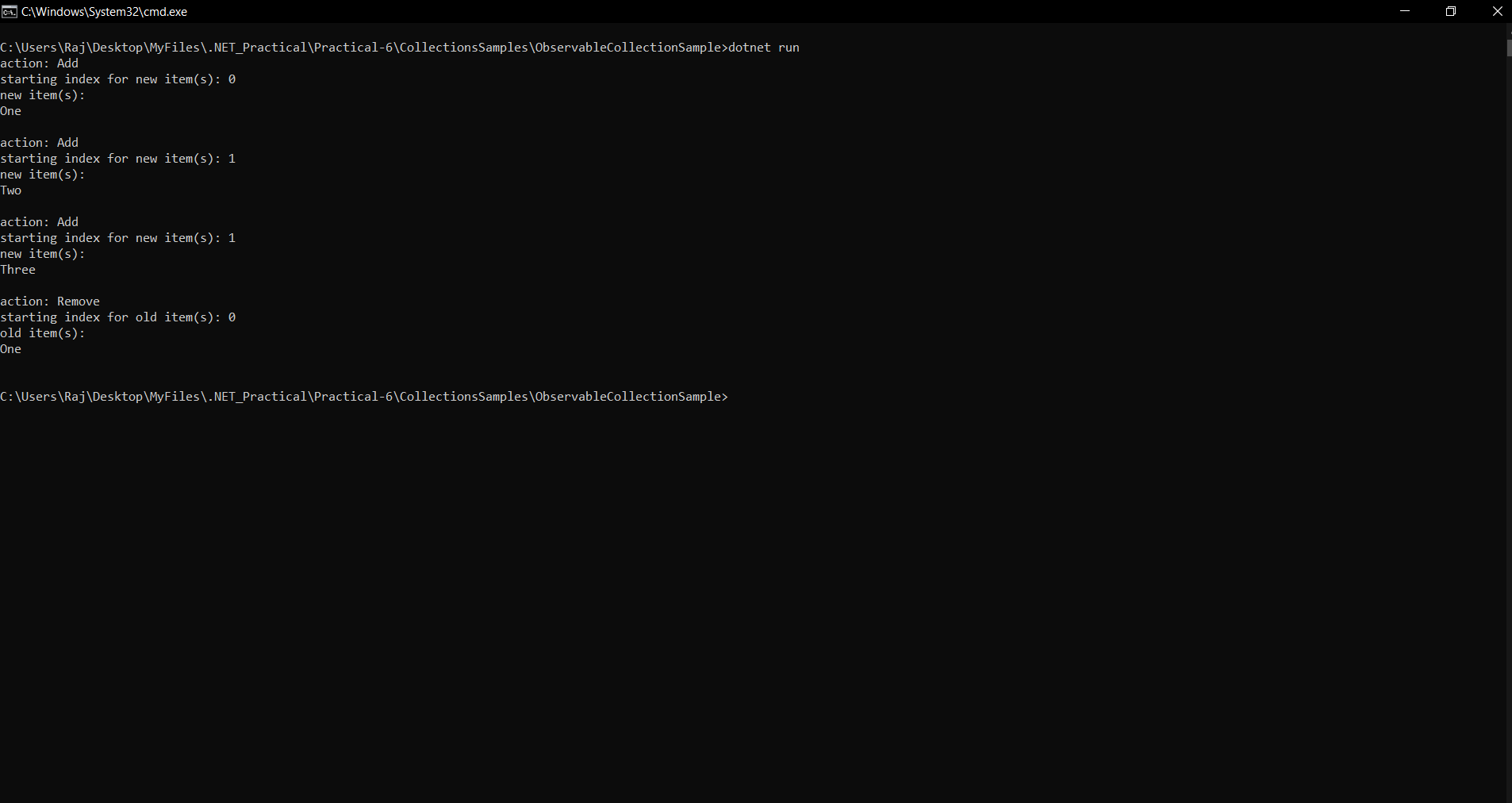
**Console.WriteLine();**

**}**

**}**

**}**

**OUTPUT:**

****

**QUEUE SAMPLE**

**CODE:**

**Document.cs**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**namespace Wrox.ProCSharp.Collections**

**{**

**public class Document**

**{**

**public string Title { get; private set; }**

**public string Content { get; private set; }**

**public Document(string title, string content)**

**{**

**this.Title = title;**

**this.Content = content;**

**}**

**}**

**}**

**DocumentManager.cs**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**public class DocumentManager**

**{**

**private readonly Queue<Document> documentQueue = new Queue<Document>();**

**public void AddDocument(Document doc)**

**{**

**lock (this)**

**{**

**documentQueue.Enqueue(doc);**

**}**

**}**

**public Document GetDocument()**

**{**

**Document doc = null;**

**lock (this)**

**{**

**doc = documentQueue.Dequeue();**

**}**

**return doc;**

**}**

**public bool IsDocumentAvailable**

**{**

**get**

**{**

**return documentQueue.Count > 0;**

**}**

**}**

**}**

**}**

**ProcessDocuments.cs**

**using System;**

**using System.Threading;**

**namespace Wrox.ProCSharp.Collections**

**{**

**public class ProcessDocuments**

**{**

**public static void Start(DocumentManager dm)**

**{**

**new Thread(new ProcessDocuments(dm).Run).Start();**

**}**

**protected ProcessDocuments(DocumentManager dm)**

**{**

**documentManager = dm;**

**}**

**private DocumentManager documentManager;**

**protected void Run()**

**{**

**while (true)**

**{**

**if (documentManager.IsDocumentAvailable)**

**{**

**Document doc = documentManager.GetDocument();**

**Console.WriteLine("Processing document {0}", doc.Title);**

**}**

**Thread.Sleep(new Random().Next(20));**

**}**

**}**

**}**

**}**

**Program.cs**

**using System;**

**using System.Threading;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var dm = new DocumentManager();**

**ProcessDocuments.Start(dm);**

**// Create documents and add them to the DocumentManager**

**for (int i = 0; i < 1000; i++)**

**{**

**Document doc = new Document("Doc " + i.ToString(), "content");**

**dm.AddDocument(doc);**

**Console.WriteLine("Added document {0}", doc.Title);**

**Thread.Sleep(new Random().Next(20));**

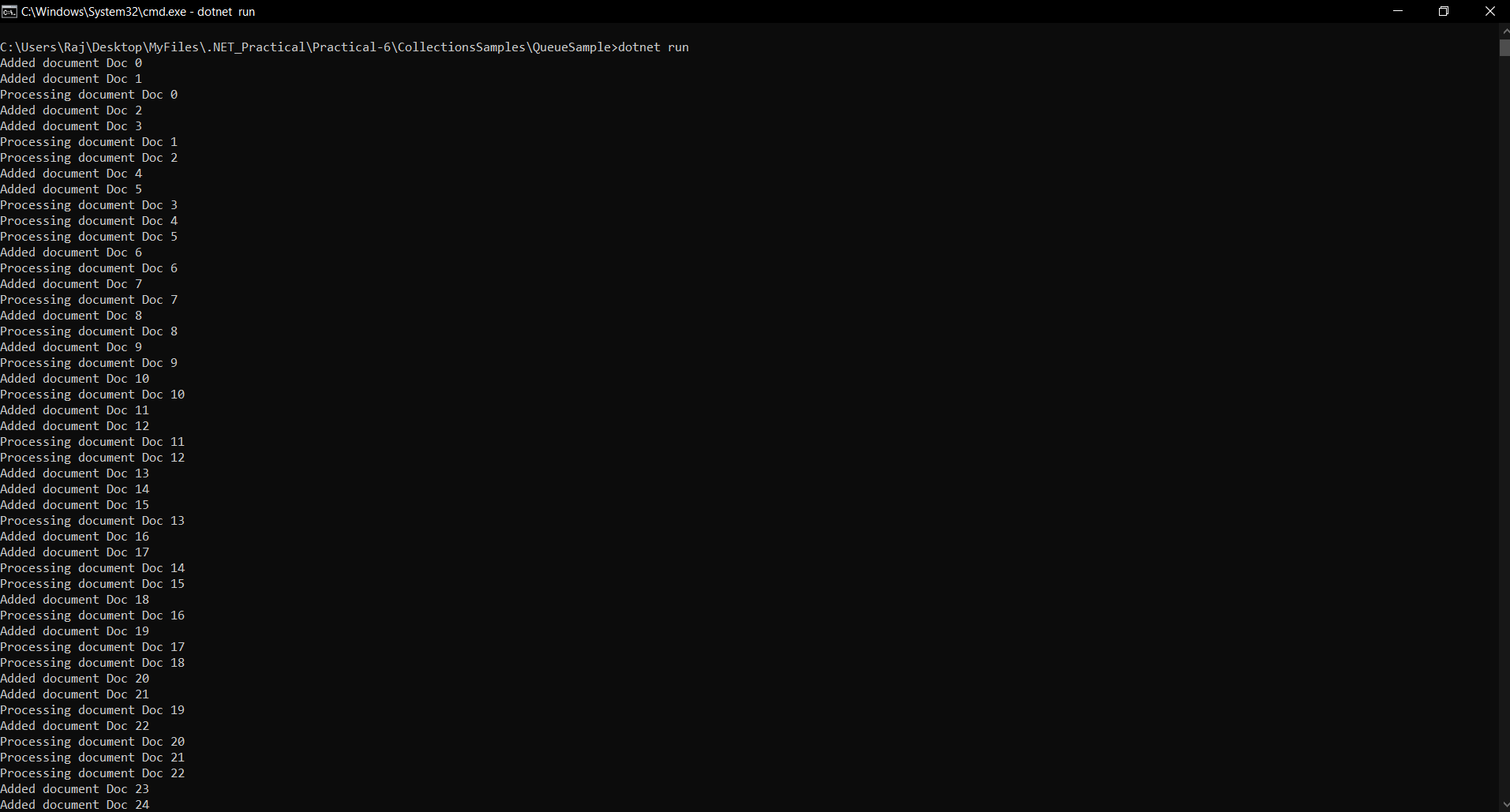
**}**

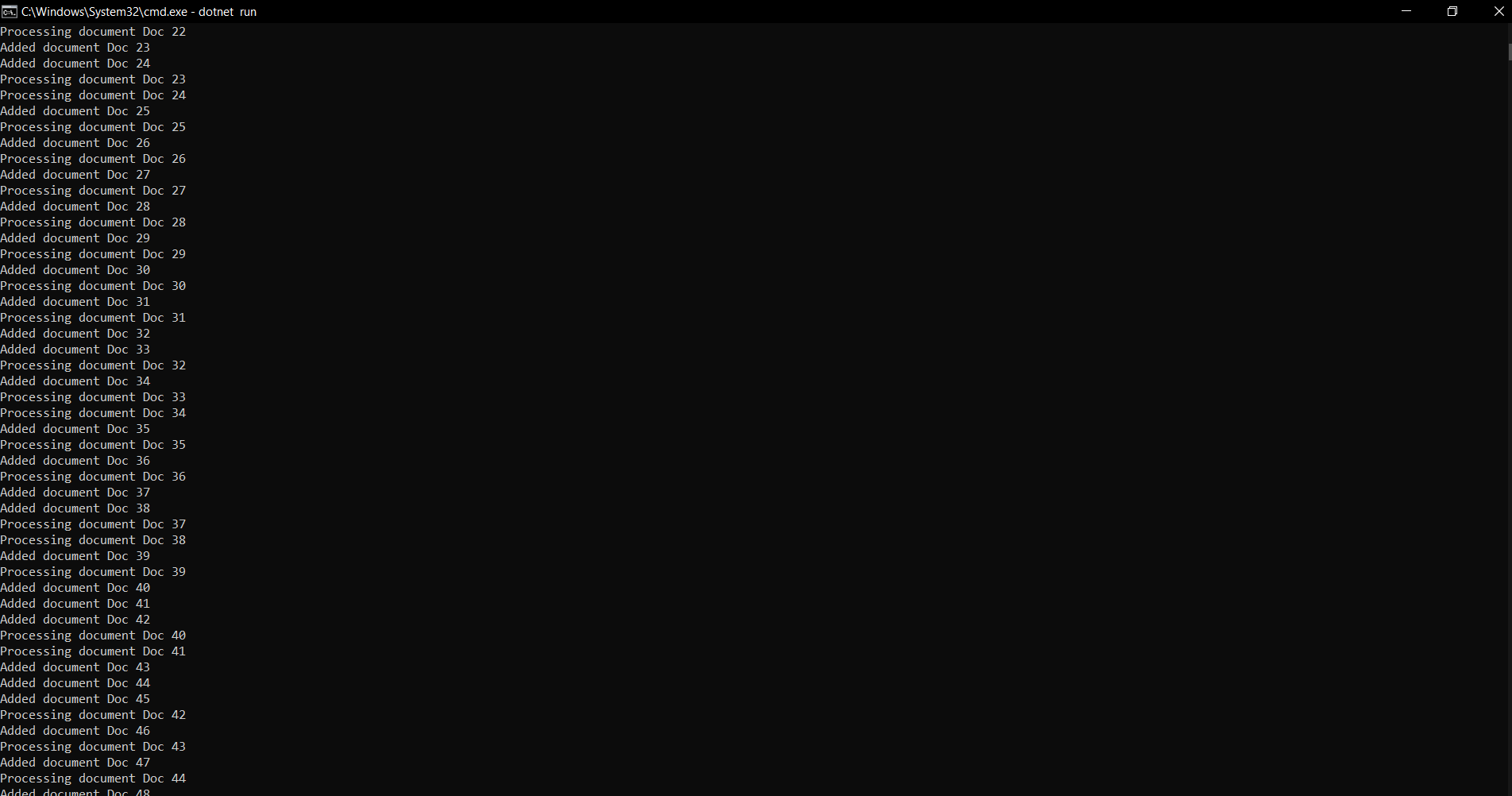
**}**

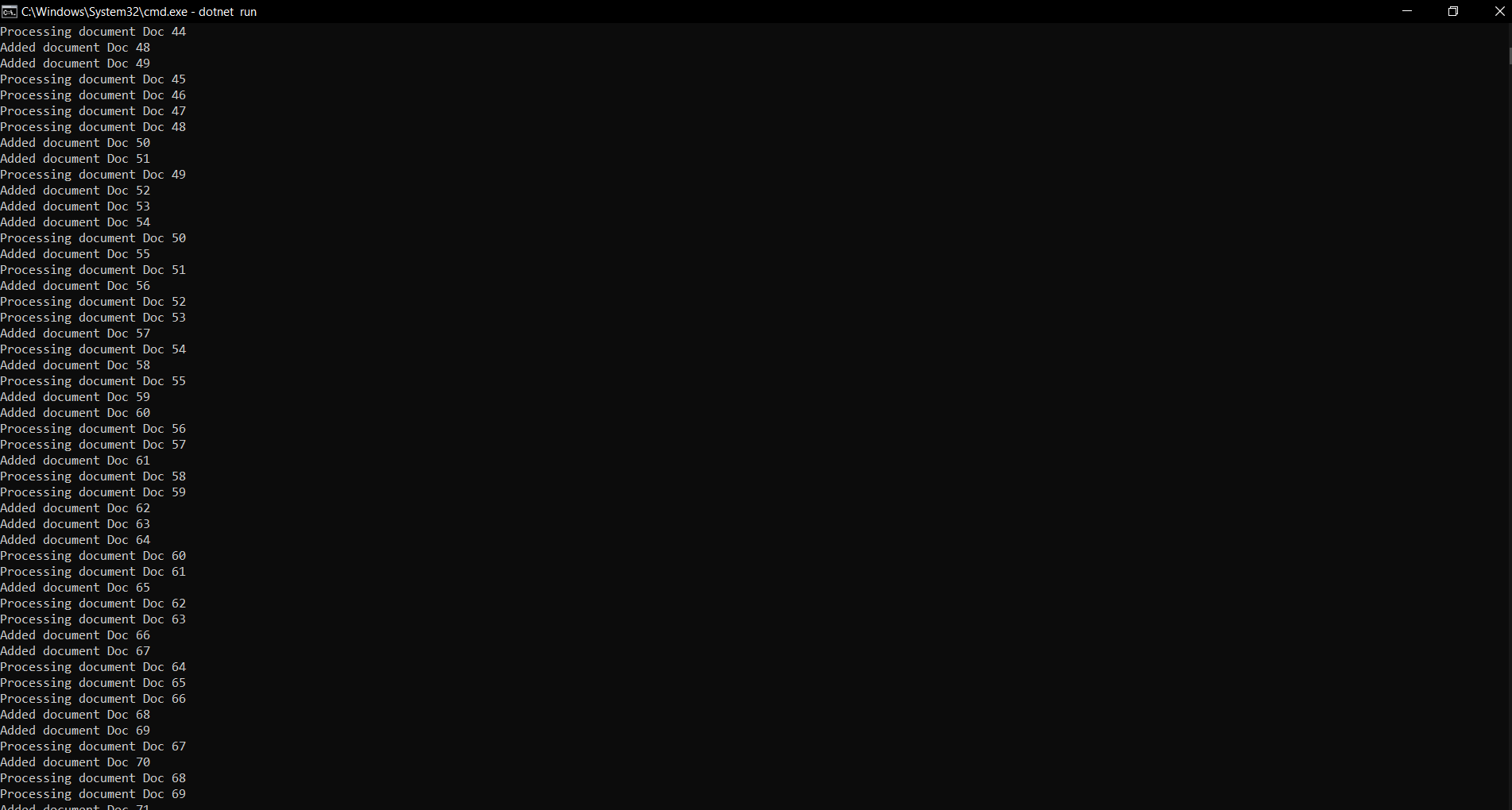
**}**

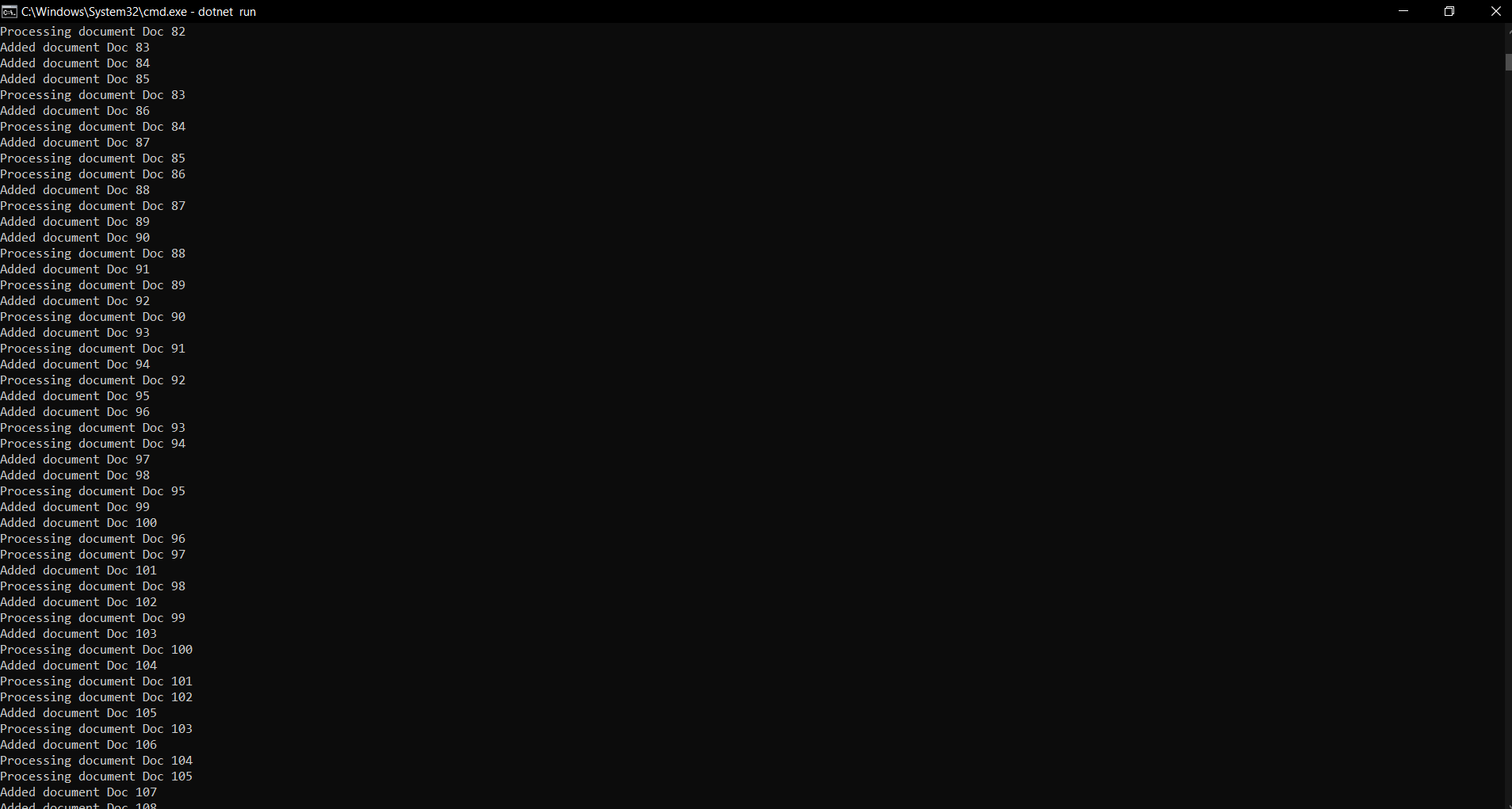
**}**

**OUTPUT:**

****

****

****

****

****

**SET SAMPLE**

**CODE:**

**using System;**

**using System.Collections.Generic;**

**namespace SetSample**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var companyTeams = new HashSet<string>() { "Ferrari", "McLaren", "Toyota", "BMW", "Renault" };**

**var traditionalTeams = new HashSet<string>() { "Ferrari", "McLaren" };**

**var privateTeams = new HashSet<string>() { "Red Bull", "Toro Rosso", "Force India", "Brawn GP" };**

**if (privateTeams.Add("Williams"))**

**Console.WriteLine("Williams added");**

**if (!companyTeams.Add("McLaren"))**

**Console.WriteLine("McLaren was already in this set");**

**if (traditionalTeams.IsSubsetOf(companyTeams))**

**{**

**Console.WriteLine("traditionalTeams is subset of companyTeams");**

**}**

**if (companyTeams.IsSupersetOf(traditionalTeams))**

**{**

**Console.WriteLine("companyTeams is a superset of traditionalTeams");**

**}**

**traditionalTeams.Add("Williams");**

**if (privateTeams.Overlaps(traditionalTeams))**

**{**

**Console.WriteLine("At least one team is the same with the traditional " +**

**"and private teams");**

**}**

**var allTeams = new SortedSet<string>(companyTeams);**

**allTeams.UnionWith(privateTeams);**

**allTeams.UnionWith(traditionalTeams);**

**Console.WriteLine();**

**Console.WriteLine("all teams");**

**foreach (var team in allTeams)**

**{**

**Console.WriteLine(team);**

**}**

**allTeams.ExceptWith(privateTeams);**

**Console.WriteLine();**

**Console.WriteLine("no private team left");**

**foreach (var team in allTeams)**

**{**

**Console.WriteLine(team);**

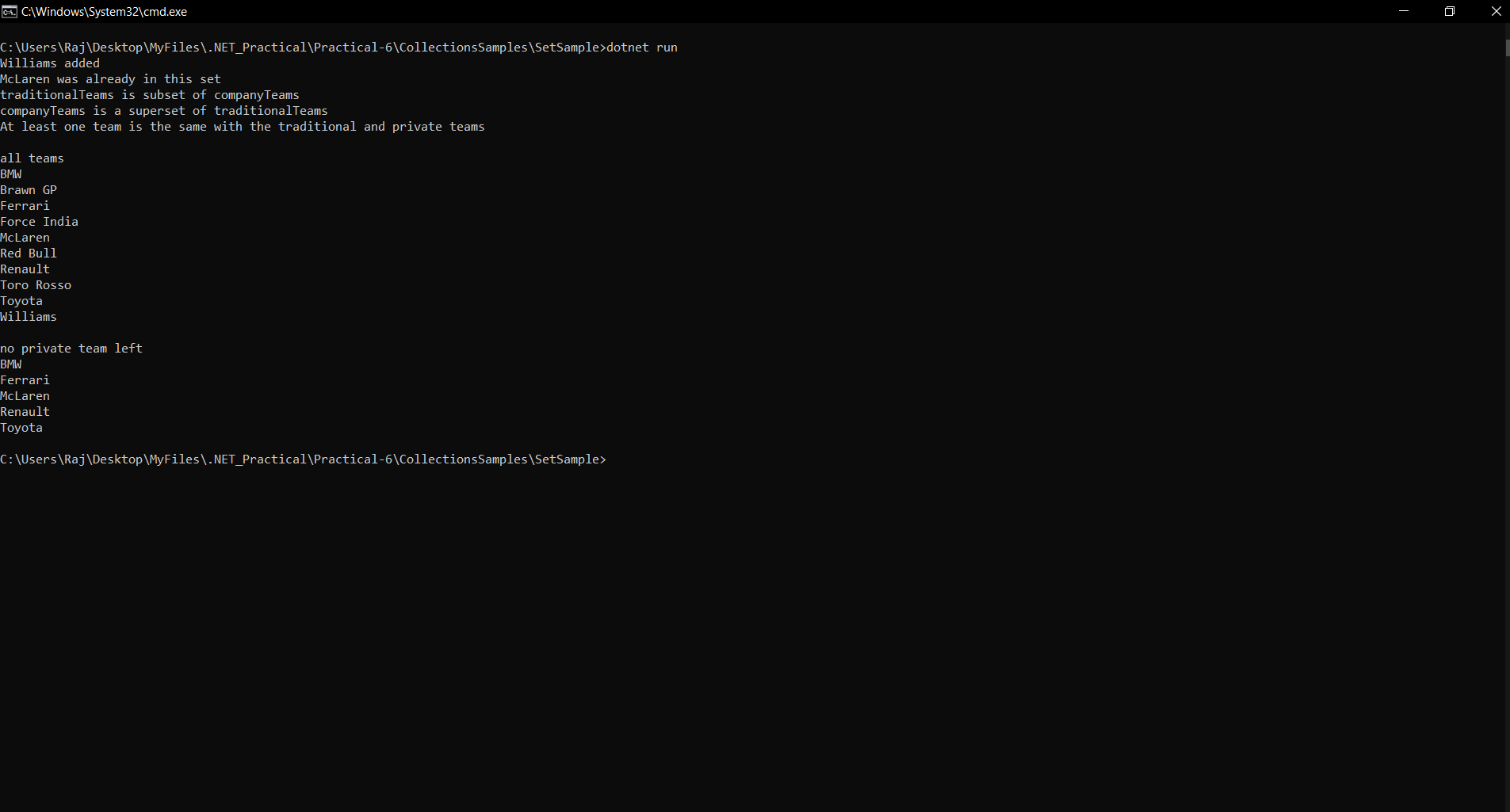
**}**

**}**

**}**

**}**

**OUTPUT:**

****

**SORTED LIST SAMPLE**

**CODE:**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**var books = new SortedList<string, string>();**

**books.Add("C# 2008 Wrox Box", "978–0–470–047205–7");**

**books.Add("Professional ASP.NET MVC 1.0", "978–0–470–38461–9");**

**books["Beginning Visual C# 2008"] = "978–0–470-19135-4";**

**books["Professional C# 2008"] = "978–0–470–19137–6";**

**foreach (KeyValuePair<string, string> book in books)**

**{**

**Console.WriteLine("{0}, {1}", book.Key, book.Value);**

**}**

**foreach (string isbn in books.Values)**

**{**

**Console.WriteLine(isbn);**

**}**

**foreach (string title in books.Keys)**

**{**

**Console.WriteLine(title);**

**}**

**{**

**string isbn;**

**string title = "Professional C# 7.0";**

**if (!books.TryGetValue(title, out isbn))**

**{**

**Console.WriteLine("{0} not found", title);**

**}**

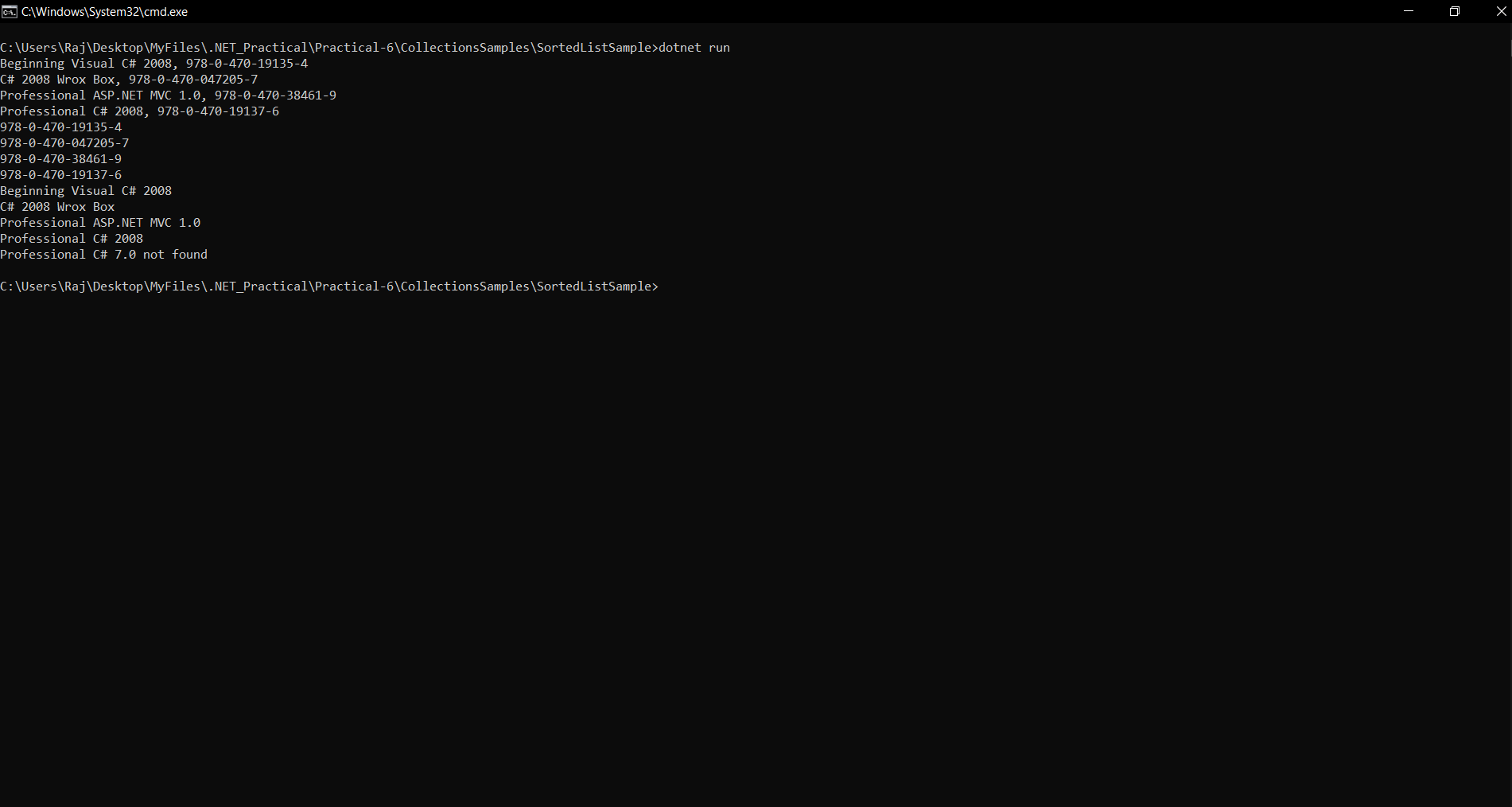
**}**

**}**

**}**

**}**

**OUTPUT:**

****

**STACK SAMPLE**

**CODE:**

**using System;**

**using System.Collections.Generic;**

**namespace Wrox.ProCSharp.Collections**

**{**

**class Program**

**{**

**static void Main()**

**{**

**var alphabet = new Stack<char>();**

**alphabet.Push('A');**

**alphabet.Push('B');**

**alphabet.Push('C');**

**Console.Write("First iteration: ");**

**foreach (char item in alphabet)**

**{**

**Console.Write(item);**

**}**

**Console.WriteLine();**

**Console.Write("Second iteration: ");**

**while (alphabet.Count > 0)**

**{**

**Console.Write(alphabet.Pop());**

**}**

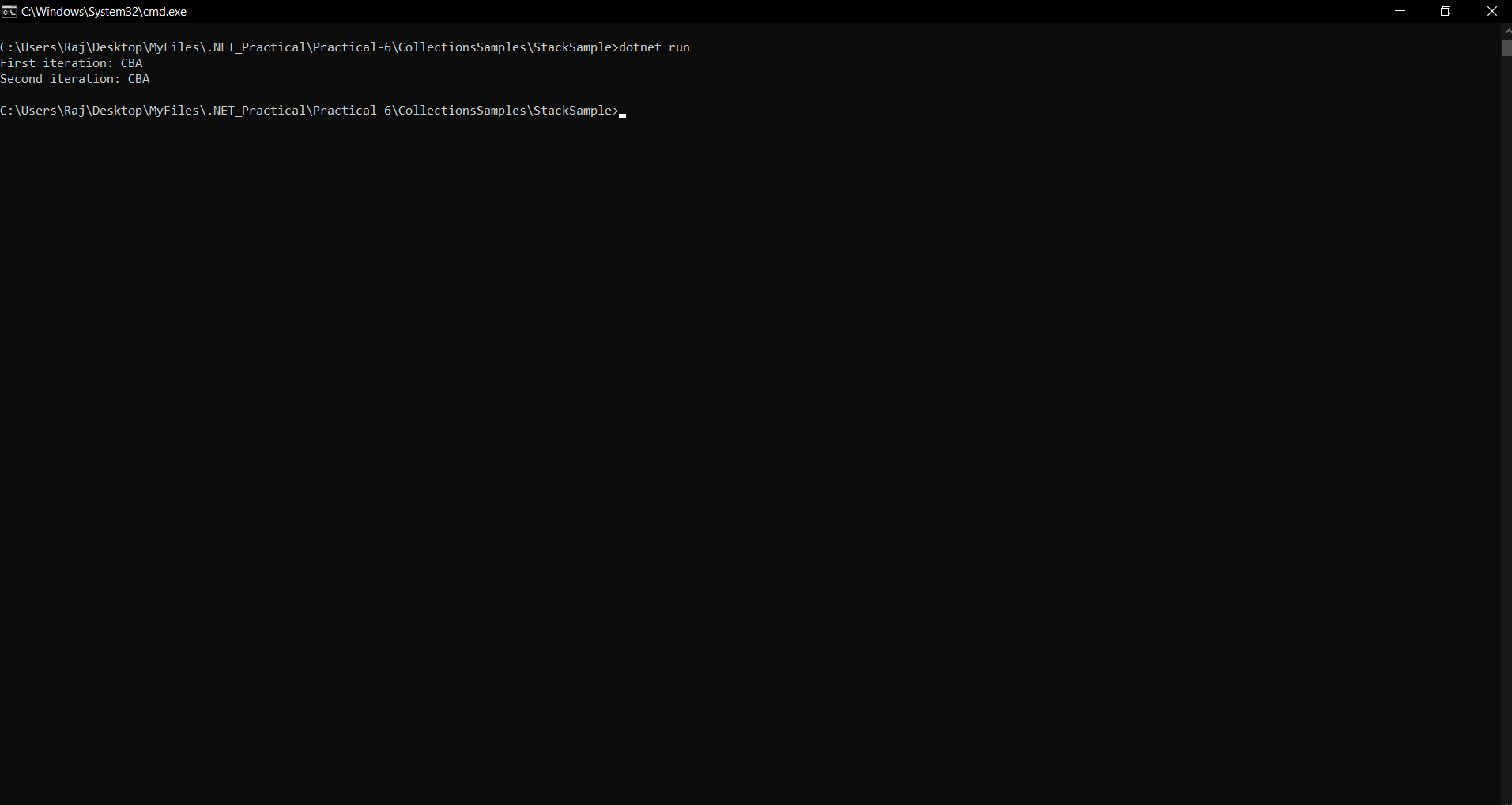
**Console.WriteLine();**

**}**

**}**

**}**

**OUTPUT:**

****