**LAB\_1 & 2**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class MsgSndEventArgs : EventArgs

{

public MsgSndEventArgs(string n)

{

this.msg = n;

}

public string getMsg()

{

return msg;

}

private string msg;

}

class Server

{

public delegate void MsgSndEventHandler(object sender, MsgSndEventArgs e);

public event MsgSndEventHandler msgSnd;

protected virtual void OnMsgSnd(MsgSndEventArgs e){

if (msgSnd != null) {

msgSnd(this, e);

}

}

//public void ClientConnect(MsgSnd msg)

//{

// msgSnd+=msg;

//}

//public void ClientDisconect(MsgSnd msg)

//{

// msgSnd -= msg;

//}

public void SendMsg(string msg)

{

OnMsgSnd(new MsgSndEventArgs(msg));

}

public void SendMsg(string msg, string client)

{

Delegate[] arr = msgSnd.GetInvocationList();

//Console.WriteLine(arr);

foreach (Delegate del in arr)

{

if (((Client)del.Target).getName().Equals(client))

{

del.DynamicInvoke(this, new MsgSndEventArgs(msg));

//del.Target(msg);

}

}

}

}

class Client

{

private string name;

public Client(string n)

{

this.name = n;

}

public string getName()

{

return name;

}

public void Connect(Server serv)

{

//Server.MsgSndEventHandler tempDelegate = new Server.MsgSndEventHandler(null, new MsgSndEventArgs(PrintMessage));

serv.msgSnd += PrintMessage;

}

public void Disconnect(Server serv)

{

//Server.MsgSnd tempDelegate = new Server.MsgSnd(PrintMessage);

//Server.MsgSndEventHandler tempDelegate = new Server.MsgSndEventHandler(null, new MsgSndEventArgs(PrintMessage));

serv.msgSnd -= PrintMessage;

}

public void PrintMessage(object sender, MsgSndEventArgs e)

{

//Console.Write("Client #" + name + " sais: ");

//Console.WriteLine(msg);

Console.Write("Client #" + name + " sais: ");

Console.WriteLine(e.getMsg());

}

}

class Program

{

static void Main(string[] args)

{

Server serv = new Server();

Client client = new Client("1");

Client client2 = new Client("2");

Client client3 = new Client("3");

client.Connect(serv);

client2.Connect(serv);

client3.Connect(serv);

serv.SendMsg("BLQ");

//Console.ReadLine();

Console.WriteLine("\*\*\*\*Sled premahvane\*\*\*\*\*");

client2.Disconnect(serv);

serv.SendMsg("Hahaha");

serv.SendMsg("aa", "3");

Console.ReadLine();

}}}

**LAB\_3 - Атрибути**

//Да се дефинира атрибут DevInfo, приложим върху клас с позиционен параметър name и именован параметър date.

//Да се приложи трикратно върху даден клас и да се изведе информация за параметритему от Main-a

#define DEBUG1

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace lab3\_zad1

{

[AttributeUsage(AttributeTargets.Struct|AttributeTargets.Enum,AllowMultiple=true)]

class MyAttributeAttribute : Attribute

{

//позиционни параметри, които са задължителни се задават като параметри на конструктора

//именованите параметри, които са незадължителни се задават като public read/write свойства

}

[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]

class MyClassAttribute : Attribute

{

public string Name { get; private set; }

public string Date { get; set; }

public MyClassAttribute(string name)

{

this.Name = name;

}

}

[MyClass("ivan", Date = "2/8/2018")]

[MyClass("maq", Date = "2/8/2018"), MyClass("toto", Date = "2/8/2018")]

class Program

{

static void Main(string[] args)

{

//Извличане

Type type = typeof(Program);

object[] classAttributes = type.GetCustomAttributes(false);

foreach (MyClassAttribute versionAttribute in classAttributes)

{

Console.WriteLine("The version of the class VersionDemo is {0}.{1}",

versionAttribute.Name, versionAttribute.Date);

}

Console.WriteLine();

foreach (object o in type.GetCustomAttributes(false))

{

MyAttributeAttribute attribute = o as MyAttributeAttribute;

if (attribute != null)

{

// attribute.

//извличане на информация чрез read свойство от attr

}

}

MyMethod(1);

Console.ReadLine();

}

[Conditional("DEBUG1")]

private static void MyMethod(int num, [CallerFilePath]string path = "", [CallerMemberName]string member = "", [CallerLineNumber]int line=0)

{

Console.WriteLine("Caller file path {0}", path);

Console.WriteLine("Caller member name {0}", member);

Console.WriteLine("Caller line number {0}", line);

}

}

}

**LAB\_4 - нишки**

//Метод с 50 итерации и 100 милисекунди забавяне на всяка итерация и разпечатване на нишката

namespace labUPR4

{

class Program

{

private static void print() {

for (int i = 0; i < 50; i++)

{

Console.WriteLine("Iter No:{0}", i);

Thread.Sleep(1);

}

}

static void Main(string[] args)

{

//Thread thrd = new Thread(new ThreadStart(print));

//thrd.Start();

Thread t = new Thread(() =>

{

for (int i = 0; i < 50; i++)

{

Console.WriteLine("Iter No:{0}", i);

Thread.Sleep(100);

}

});

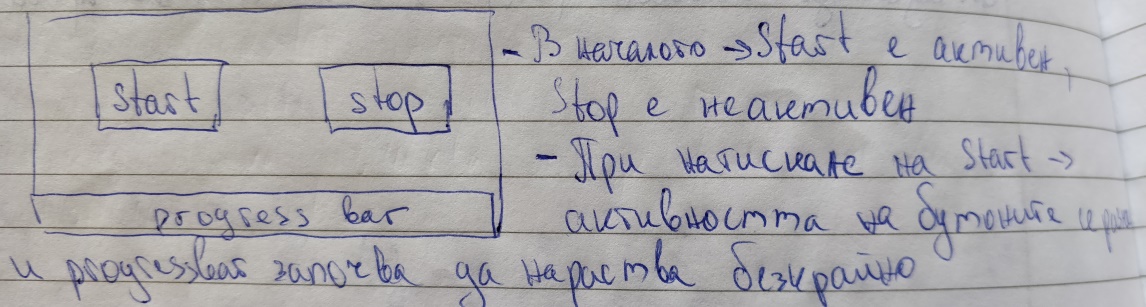
t.Start();

Console.ReadLine();

}

}

}



namespace nishki

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

static int countReached = 0;

static bool thrdWork = false;

Thread thrd;

private void PBUpdate()

{

try

{

while (thrdWork)

{

//do your work;

for (int i = 0; i < 100; i++)

{

progressBar1.Invoke(new Action(() =>

{

if (progressBar1.Value < 100)

{

progressBar1.Value += 1;

}

else

{

progressBar1.Value = 0;

}

}));

Thread.Sleep(100);

}

countReached += 1;

}

}

catch {

progressBar1.Invoke(new Action(() =>

{

progressBar1.Value = 0;

}));

}

}

private void btnStart\_Click(object sender, EventArgs e)

{

btnStop.Enabled = true;

btnStart.Enabled = false;

thrd = new Thread(new ThreadStart(PBUpdate));

thrdWork = true;

thrd.Start();

}

private void btnStop\_Click(object sender, EventArgs e)

{

if (countReached < 1)

{

return;

}

else {

thrd.Abort();

btnStop.Enabled = false;

btnStart.Enabled = true;

thrdWork = false;

countReached = 0;

}

}

private void Form1\_Load(object sender, EventArgs e)

{}

}

}

LAB\_5 - synchronisation

namespace labUpr5 {

interface MyCounter {

void Increment();

void Decrement();

int Value {

get;

}

}

/// <summary>

/// no sync

/// </summary>

class MyCounterRaw : MyCounter {

int value;

public void Increment() {

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

}

public void Decrement() {

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

}

public int Value {

get { return value; }

}

}

/// <summary>

/// Synchronization

/// </summary>

[Synchronization]

class MyCounterSync1 : ContextBoundObject, MyCounter {

int value;

public void Increment() {

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

}

public void Decrement() {

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

}

public int Value {

get { return value; }

}

}

/// <summary>

/// Monitor

/// </summary>

class MyCounterSync2 : MyCounter {

int value;

public void Increment() {

Monitor.Enter(this);

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

Monitor.Exit(this);

}

public void Decrement() {

Monitor.Enter(this);

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

Monitor.Exit(this);

}

public int Value {

get { return value; }

}

}

/// <summary>

/// MethodImplOptions.Synchronized

/// </summary>

class MyCounterSync3 : MyCounter {

int value;

[MethodImpl(MethodImplOptions.Synchronized)]

public void Increment() {

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

}

[MethodImpl(MethodImplOptions.Synchronized)]

public void Decrement() {

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

}

public int Value {

get { return value; }

}

}

/// <summary>

/// Mutex

/// </summary>

class MyCounterSync4 : MyCounter {

int value;

Mutex m = new Mutex(false);

public void Increment() {

m.WaitOne();

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

m.ReleaseMutex();

}

public void Decrement() {

m.WaitOne();

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

m.ReleaseMutex();

}

public int Value {

get { return value; }

}

}

/// <summary>

/// Semaphore

/// </summary>

class MyCounterSync5 : MyCounter {

int value;

Semaphore s = new Semaphore(1, 1);

public void Increment() {

s.WaitOne();

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

s.Release();

}

public void Decrement() {

s.WaitOne();

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

s.Release();

}

public int Value {

get { return value; }

}

}

/// <summary>

/// AutoResetEvent

/// </summary>

class MyCounterSync6 : MyCounter {

int value;

AutoResetEvent e = new AutoResetEvent(true);

public void Increment() {

e.WaitOne();

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

e.Set();

}

public void Decrement() {

e.WaitOne();

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

e.Set();

}

public int Value {

get { return value; }

}

}

/// <summary>

/// lock

/// </summary>

class MyCounterSync7 : MyCounter {

int value;

object locker = new object();

public void Increment() {

lock (locker) {

int temp = value;

Thread.Sleep(20);

temp++;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Incremented from:" + temp + " to:" + value);

}

}

public void Decrement() {

lock (locker) {

int temp = value;

Thread.Sleep(20);

temp--;

Thread.Sleep(20);

value = temp;

Console.WriteLine("Thread ID: " + Thread.CurrentThread.ManagedThreadId + " -> Decremented from:" + temp + " to:" + value);

}

}

public int Value {

get { return value; }

}

}

class Program {

static void Main(string[] args) {

List<MyCounter> MyCounter = new List<MyCounter>();

MyCounter.Add(new MyCounterRaw());

MyCounter.Add(new MyCounterSync1());

MyCounter.Add(new MyCounterSync2());

MyCounter.Add(new MyCounterSync3());

MyCounter.Add(new MyCounterSync4());

MyCounter.Add(new MyCounterSync5());

MyCounter.Add(new MyCounterSync6());

MyCounter.Add(new MyCounterSync7());

int[] results = new int[MyCounter.Count];

for (int i = 0; i < MyCounter.Count; i++) {

results[i] = TestThisObject(MyCounter[i]);

}

for (int i = 0; i < MyCounter.Count; i++) {

Console.WriteLine("Sync Method #" + i + " returned:" + results[i]);

}

Console.ReadKey();

}

private static int TestThisObject(MyCounter obj) {

List<Thread> threads = new List<Thread>();

for (int i = 0; i < 20; i++) {

Thread t;

if (i % 2 == 0) {

//t = new Thread(() => { for (int j = 0; j < 5; j++) obj.Increment(); });

t = new Thread(() => { obj.Increment(); });

} else {

//t = new Thread(() => { for (int j = 0; j < 5; j++) obj.Decrement(); });

t = new Thread(() => { obj.Decrement(); });

}

t.Start();

threads.Add(t);

Console.WriteLine("Started thread {0} ", i);

Thread.Sleep(2);

}

for (int i = 0; i < 20; i++) {

threads[i].Join();

}

Console.WriteLine("Final value = {0} \n\*\*\*\*\*\*\*\*\*\*\*\*\*\*", obj.Value);

return obj.Value;

}

}

}

LAB\_6 - асинхронен метод

//Асинхронен метод

// - sleep в sec

// - връща ID на текущата нишка в ret параметъра

// - връща като резултат символна нишка

namespace labUpr6 {

class Program {

delegate string MyDelegate(ref int a, int b);

static string sleep(ref int thrdID, int sleepMS)

{

Thread.Sleep(sleepMS);

thrdID = Thread.CurrentThread.GetHashCode();

return "Exit from async method.";

}

static void MyCallback2(IAsyncResult ar) {

int thrdID = -1;

string res = ((MyDelegate)(((AsyncResult)ar).AsyncDelegate)).EndInvoke(ref thrdID, ar);

}

static void MyCallback3(IAsyncResult ar)

{

int thrdID = -1;

string res = ((MyDelegate)(((AsyncResult)ar).AsyncDelegate)).EndInvoke(ref thrdID, ar);

Console.WriteLine("ThrdID : {0}, and return message is {1}", thrdID, res);

}

static void Main(string[] args) {

//first method

int thrdID1 = -1;

MyDelegate d1 = new MyDelegate(sleep);

IAsyncResult ar1 = d1.BeginInvoke(ref thrdID1, 1000, null, null);

string endInvoke1 = d1.EndInvoke(ref thrdID1, ar1);

Console.WriteLine("first method - > ThrdID : {0}, and return message is {1}", thrdID1, endInvoke1);

//second method

int thrdID2 = -1;

MyDelegate d2 = new MyDelegate(sleep);

object state2 = new object();

IAsyncResult ar2 = d2.BeginInvoke(ref thrdID2, 1000, MyCallback2, state2);

string endInvoke2 = d2.EndInvoke(ref thrdID2, ar2);

Console.WriteLine("second method - > ThrdID : {0}, and return message is {1}", thrdID2, endInvoke2);

//third method

int thrdID3 = -1;

MyDelegate d3 = new MyDelegate(sleep);

IAsyncResult ar3 = d3.BeginInvoke(ref thrdID3, 1000, MyCallback3, d3);

//string endInvoke3 = ((MyDelegate)ar.AsyncState).EndInvoke(ref thrdID, ar);

Console.ReadLine();

}

}

}

LAB\_7 – Task Parallel Library

namespace labUPR7

{

class Program

{

static CancellationTokenSource source;

private static void print() {

CancellationToken token = source.Token;

token.Register(new Action(end));

Console.WriteLine("Task Started.");

for (int i = 0; i < 10000; i++) {

Thread.Sleep(1000);

if (token.IsCancellationRequested)

{

break;

}

}

}

private static void end()

{

Console.WriteLine("Task ending.");

for (int i = 0; i < 1; i++)

{

Thread.Sleep(1000);

}

}

static void Main(string[] args)

{

source = new CancellationTokenSource();

Task task = new Task(new Action(print));

task.Start();

Thread.Sleep(3000);

source.Cancel();

Console.WriteLine("Test");

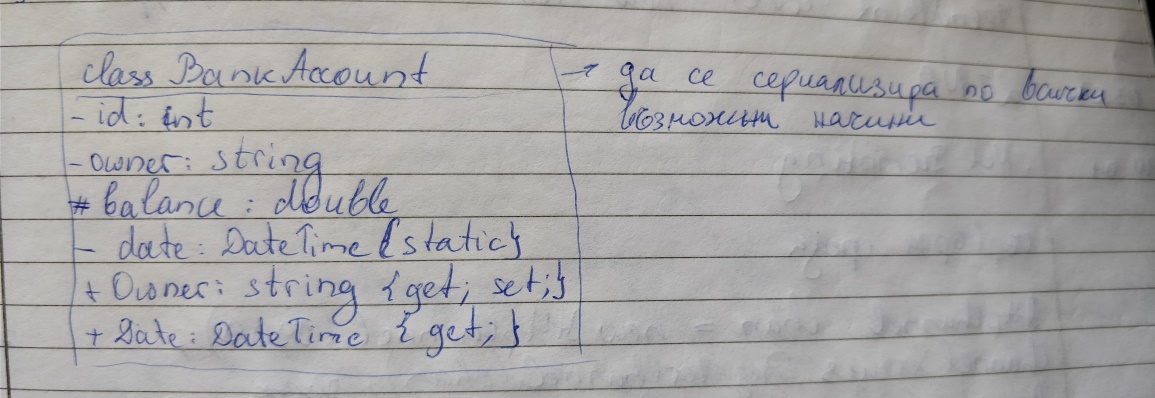
Console.ReadLine();

}

}

}

**LAB\_8 – Serialization**



namespace labUpr8

{

[DataContract]

[Serializable]

public class BankAccount : ISerializable

{

[DataMember]

private int id;

[DataMember]

private string owner;

[DataMember]

protected double balance;

[DataMember]

private static DateTime date;

public void GetObjectData(SerializationInfo info, StreamingContext sc) {

info.AddValue("Owner", owner);

info.AddValue("Id", id);

info.AddValue("Balance", balance);

info.AddValue("Date", date);

}

public BankAccount(SerializationInfo info, StreamingContext st)

{

owner = (string) info.GetValue("Owner",typeof(string));

id = (int) info.GetValue("Id",typeof(int));

balance = (double)info.GetValue("Balance", typeof(double));

date = (DateTime)info.GetValue("Date", typeof(DateTime));

}

public BankAccount()

{

Random rnd = new Random();

owner = "Jon Doe";

id = rnd.Next(100000000);

balance = 0.0;

date = DateTime.Now;

}

public BankAccount(string ownr)

{

Random rnd = new Random();

owner = ownr;

id = rnd.Next(100000000);

balance = 0.0;

date = DateTime.Now;

}

public string Owner

{

get { return owner; }

set { owner = value; }

}

public DateTime Date

{

get { return date; }

}

}

public class Program

{//soap, date contract, custom

static void Main(string[] args)

{

//BINARY

BankAccount p = new BankAccount("first");

Stream s = File.Open("BankAccountBinary", FileMode.Create);

BinaryFormatter fmt = new BinaryFormatter();

fmt.Serialize(s, p);

s.Close();

Stream r = File.Open("BankAccountBinary", FileMode.Open);

BankAccount restored = (BankAccount)fmt.Deserialize(r);

r.Close();

//BINARY

//XML

BankAccount pXML = new BankAccount("second");

System.Xml.Serialization.XmlSerializer xml = new System.Xml.Serialization.XmlSerializer(typeof(BankAccount));

Stream sXML = File.Open("BankAccountXML", FileMode.Create);

xml.Serialize(sXML, pXML);

sXML.Close();

Stream rXML = File.Open("BankAccountXML", FileMode.Open);

BankAccount restoredXML = (BankAccount)xml.Deserialize(rXML);

rXML.Close();

//XML

////DATA CONTRACT // ne raboti ako klasa bank akaunt e nasledil ISerializable, zashtoto stava konflikt s DataContract

//BankAccount pDC = new BankAccount("third");

//DataContractSerializer sDC = new DataContractSerializer(typeof(BankAccount));

//XmlWriter wDC = XmlWriter.Create("BankAccountDC");

//sDC.WriteObject(wDC, pDC);

//wDC.Close();

//XmlReader rDC = XmlReader.Create("BankAccountDC");

//BankAccount restoredDC = (BankAccount)sDC.ReadObject(rDC, false);

//wDC.Close();

////DATA CONTRACT

//SOAP

BankAccount pSoap = new BankAccount("fourth");

Stream sSoap = File.Open("BankAccountSoap", FileMode.Create);

SoapFormatter fmtSoap = new SoapFormatter();

fmtSoap.Serialize(sSoap, pSoap);

sSoap.Close();

Stream rSoap = File.Open("BankAccountSoap", FileMode.Open);

BankAccount restoredSoap = (BankAccount)fmtSoap.Deserialize(rSoap);

r.Close();

//SOAP

//CUSTOM

BinaryFormatter fmtCust = new BinaryFormatter();

BankAccount pCust = new BankAccount("fifth");

// Serialize an object

using (var streamCust = File.OpenWrite("BankAccountCust"))

{

fmtCust.Serialize(streamCust, pCust);

}

// Deserialzie it from the file

BankAccount restoredCust = new BankAccount();

using (var streamCust = File.OpenRead("BankAccountCust"))

{

restoredCust = (BankAccount)fmtCust.Deserialize(streamCust);

}

//CUSTOM

Console.ReadLine();

}

}

}

**LAB\_9 – Sockets**

namespace labupr9

{

class ServerProgram

{

private static List<string> onlineClients = new List<string>();

List<Thread> connections = new List<Thread>();

static void Main(string[] args)

{

//IPHostEntry host = Dns.GetHostEntry("localhost");

//foreach (IPAddress addr in host.AddressList) {

// Console.WriteLine("Address : {0}",addr.ToString());

//}

IPAddress servAddr = IPAddress.Parse("127.0.0.1");

int portNo = 9999;

IPEndPoint servEP = new IPEndPoint(servAddr, portNo);

Socket servSock = new Socket(

AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp

);

servSock.Bind(servEP);

servSock.Listen(10);

for (; ; ) {

handleClient(servSock.Accept());

}

servSock.Close();

}

public static void handleClient(Socket clientSock)

{

//comm

string message;

Console.WriteLine("[" + DateTime.Now.ToString() + "] Joined client : {0}", clientSock.RemoteEndPoint.ToString());

do

{

byte[] msg = new byte[1024];

clientSock.Receive(msg);

message = Encoding.ASCII.GetString(msg);

Console.WriteLine("[" + DateTime.Now.ToString() + "] Message from {0} : {1}", clientSock.RemoteEndPoint.ToString(), message.Remove(' '));

Console.Write("[" + DateTime.Now.ToString() + "] Server response : ");

message = Console.ReadLine();

msg = Encoding.ASCII.GetBytes(message);

clientSock.Send(msg);

} while (!message.ToLower().Equals("exit"));

onlineClients.Remove(clientSock.RemoteEndPoint.ToString());

clientSock.Shutdown(SocketShutdown.Both);//samo za klientski soketi

clientSock.Close();

}

}

}

**LAB\_9 – CLIENT Sockets**

namespace labupr9Client

{

class ClientProgram

{

static void Main(string[] args)

{

System.Net.IPAddress servAddr = IPAddress.Parse("127.0.0.1");

int portNo = 9999;

IPEndPoint servEP = new IPEndPoint(servAddr, portNo);

Socket clientSock = new Socket(

AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp

);

clientSock.Connect(servEP);

//send/recieve

byte[] msgSend = new byte[1024];

byte[] msgRec = new byte[1024];

string messageToSend;

do

{

Console.Write("[" + DateTime.Now.ToString() + "] Message to send to server : ");

messageToSend = Console.ReadLine();

msgSend = Encoding.ASCII.GetBytes(messageToSend);

clientSock.Send(msgSend);

clientSock.Receive(msgRec);

Console.WriteLine("[" + DateTime.Now.ToString() + "] Server : " + Encoding.ASCII.GetString(msgRec).Remove(' '));

} while (!messageToSend.ToLower().Equals("exit"));

clientSock.Shutdown(SocketShutdown.Both);

clientSock.Close();

Console.ReadLine();

}

}

}

**LAB\_10 – ?**

namespace labupr10

{

class Program

{

static void Main(string[] args)

{

int[] nums = { 1, 4, -2, 5, -6 };

IEnumerable<int> res = from n in nums

where n < 0

select n;

printRes(res);

nums[1] = -8;

printRes(res);

List<Person> persons = new List<Person>();

persons.Add(new Person("Prsn1",1));

persons.Add(new Person("Prsn2",2));

persons.Add(new Person("Prsn3",3));

var res1 = from p in persons

where p.Id > 1 & p.Id < 3

select p.Name; //new {p.Name} // -> anonimen tip

printRes(res1);

Console.WriteLine(persons[0].GetFamily()); // extension method, seen with a down arrow

//var newRes = from p in persons

// join s in student on p.Id equals s.Id into ps

// from c in citizens

// where ...

// select new {};

XElement elem = new XElement("Person", "Ivan");

Console.WriteLine(elem);

Console.WriteLine((string)elem);

//XElement personsXML = new XElement("Persons",

//new XElement("Person",

// new XElement("Name", persons[0].Name),

// new XAttribute("Id", persons[0].Id)),

// new XElement("Person",

// new XElement("Name", persons[1].Name),

// new XAttribute("Id", persons[1].Id)),

// new XElement("Person",

// new XElement("Name", persons[2].Name),

// new XAttribute("Id", persons[2].Id)));

XStreamingElement personsXML = new XStreamingElement("Persons",

from pers in persons select new XElement("Person",

new XElement("Name", pers.Name),

new XAttribute("Id", pers.Id)));

XDocument doc = new XDocument(personsXML);

doc.Declaration = new XDeclaration("1.0", "utf-8", "yes");

Console.WriteLine(doc);

persons[0].Name = "Pers11";

Console.WriteLine(doc);

doc.Save("Persons.xml");

XDocument doc1 = XDocument.Load("Persons.xml");

var persons123 = from p in doc1.Descendants("Person")

select new Person

{

Name = (string)p.Element("Name"),

Id = (int)p.Attribute("Id")

};

Console.WriteLine("TEST "+persons123);

Console.ReadLine();

}

static void printRes(IEnumerable<int> res) {

foreach (int num in res)

{

Console.WriteLine("Negative number : {0}", num);

}

}

static void printRes(IEnumerable<string> res)

{

foreach (string name in res)

{

Console.WriteLine("{0} is with id = 2", name);

}

}

}

class Person

{

public string Name { get; set; }

public int Id { get; set; }

public Person(string name, int id) {

Name = name;

Id = id;

}

public Person() {

Name = "Jon";

Id = -1;

}

}

static class MyClass {

public static string GetFamily (this Person p){

return p.Name;

}

}

}

**LAB\_11 – .Net Remoting**

**namespace ClassLibrary1\_lab11**

{

public interface IMyInterface

{

int RetInt();

}

}

**namespace labb11**

{

class MyClass : MarshalByRefObject, IMyInterface

{

static int num = 0;

public int RetInt()

{

num++;

Console.WriteLine("Num {0}:", num);

return num;

}

}

class Program

{

static void Main(string[] args)

{

HttpChannel chan = new HttpChannel(9999);

ChannelServices.RegisterChannel(chan,false);

RemotingConfiguration.RegisterWellKnownServiceType(typeof(MyClass), "MyObj", WellKnownObjectMode.SingleCall);

Console.ReadKey();

}

}

}

**namespace labb11\_client**

{

class Program

{

static void Main(string[] args)

{

HttpChannel chan = new HttpChannel();

ChannelServices.RegisterChannel(chan, false);

RemotingConfiguration.RegisterWellKnownServiceType(typeof(string), "MyObj", WellKnownObjectMode.SingleCall);

IMyInterface obj = (IMyInterface)Activator.GetObject(typeof(IMyInterface),"http://localhost:9999/MyObj");

Console.WriteLine("Client: " + obj.RetInt());

Console.ReadLine();

}

}

}

**namespace labb11\_order\_server**

{

class MyClass : MarshalByRefObject, IMyInterface

{

static int num = 0;

public int RetInt()

{

num++;

Console.WriteLine("Num {0}:", num);

return num;

}

}

class Program

{

static void Main(string[] args)

{

HttpChannel chan = new HttpChannel(9999);

ChannelServices.RegisterChannel(chan,false);

RemotingConfiguration.RegisterWellKnownServiceType(typeof(MyClass), "MyObj", WellKnownObjectMode.SingleCall);

object[] url = { new UrlAttribute("tcp://localhost:1234/Server") };

string obj = (string)Activator.CreateInstance(

typeof(string),

"asd",

url);

Console.ReadKey();

} }}