**Einsendeaufgaben zu CSH03**

|  |  |  |
| --- | --- | --- |
| Name: Zang | Vorname:Keping | **Ihr Fernlehrer:** |
| Postleitzahl und Ort:73760 | Straße:Scharnhauser Straße | **Senkbeil** |
| Studien-Nr.:793302 | Lehrgangs-Nr.:460 | Note: gut (2) |
| Einsendeaufgabe: **CSH03-XX** | Auflage: **2** |  |

Füllen Sie das Adressfeld (*die nicht hinterlegten Felder*) bitte sorgfältig aus.

|  |  |  |
| --- | --- | --- |
| **Nr.** | **Aufgaben/Lösung** | **Punkte** |
|  | Lösen Sie bitte die Einsendeaufgaben und senden Sie Ihre Lösung per Post oder E-Mail über das Online-Lernsystem an uns ein. |  |
|  | *Vorbemerkung: Der letzte Bearbeitungszustand des Fliegerprojekts (Lektion 5) wird in diesen Einsendeaufagben benötigt und zur Kontrolle auch im Online--System zu diesem Lernelement bereitgestellt. Das betrifft die beiden init-Dateien für die Flüge „LH 3000“ und „LH 500“ sowie die Projektdatei „Fliegerprojekt\_ESA\_CSH03.cs“.* |  |
| 1. | a) Wie öffnen Sie eine vorhandene Datei, sodass deren Inhalt gelöscht wird, damit sodann neue Inhalte in die Datei geschrieben werden können?  Man kann „Create“ Methode benutzen. Die Methode “Create“ wird eine neue Datei erstellen. Wenn die Datei noch nicht existiert, wird eine neue Datei erstellt. Wenn die Datei schon existiert, wird das Inhalt der Datei gelöscht werden, dann werden neue Inhalte in die Datei geschrieben .  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  namespace Hausaufgabe1  {  class Program  {  static void Main(string[] args)  {  StreamWriter write = new StreamWriter(File.Open(@"D:\Dokumente\Ping\fernstudium\Hausaufgabe1.txt", FileMode.Create));  write.WriteLine("hello world!");  write.Close();  }  }  }  Die neue txt Datei „Hausaufgabe1“ mit Inhalte „Hello world!“ ist erstellt.  Jetzt ändere ich die Inhalte  static void Main(string[] args)  {  StreamWriter write = new StreamWriter(File.Open(@"D:\Dokumente\Ping\fernstudium\Hausaufgabe1.txt", FileMode.Create));  write.WriteLine("Aufgabe1");  write.Close();  }  Die Datei“Hausaufgabe1“ wird jetzt mit Inhalte“Aufgabe1“ erstellt.  Hier ist der Filemode „Truncate“ zu wählen. |  |
| b) Wie stellen Sie sicher, dass eine Datei geöffnet wird, falls sie existiert, und anderenfalls neu erstellt wird?  ... Man kann das Mitglied „OpenOrCreate“ der FileMode Enumeration benutzen. Mit dem Mitglied „OpenOrCreate“ wird eine Datei geöffnet, wenn die Datei existiert . Wenn die Datei nicht existiert, wird eine neue Datei erstellt.  namespace Hausaufgabe1  {  class Program  {  static void Main(string[] args)  {  StreamWriter write = new StreamWriter(File.Open(@"D:\Dokumente\Ping\fernstudium\Hausaufgabe1.txt", FileMode.OpenOrCreate));  write.WriteLine("Aufgabe1");  write.Close();  }  }  } //Datei“Hausaufgabe1“ wird geöffnet.  namespace Hausaufgabe1  {  class Program  {  static void Main(string[] args)  {  StreamWriter write = new StreamWriter(File.Open(@"D:\Dokumente\Ping\fernstudium\Hausaufgabe2.txt", FileMode.OpenOrCreate));  write.WriteLine("Aufgabe2");  write.Close();  }  }  }//Eine neue Datei“Hausaufgabe2“ wird mit Inhalt“Aufgabe2“ erstellt.  Korrekt gelöst! |  |
| Nennen Sie jeweils kurz den Lösungsansatz und geben Sie ein Codierungsbeispiel.  Punkte: 5/10 |  |
| 2. | Gegeben sei das folgende Byte-Array:  byte[] array = { 32, 32, 67, 67, 32, 32, 32, 35, 32, 35, 32,                   32, 67, 32, 32, 67, 32, 32, 35, 32, 35, 32,                   67, 32, 32, 32, 32, 32, 35, 35, 35, 35, 35,                   67, 32, 32, 32, 32, 32, 32, 35, 32, 35, 32,                   67, 32, 32, 32, 32, 32, 35, 35, 35, 35, 35,                   32, 67, 32, 32, 67, 32, 32, 35, 32, 35, 32,                   32, 32, 67, 67, 32, 32, 32, 35, 32, 35, 32 };  Schreiben Sie eine Methode „ESA2In“ in einem neuen Projekt „ESA\_2“, mit der dieses Array in eine in Ihrem Projektmappenordner anzulegende Datei „ESA2.txt“ gespeichert werden kann.  Schreiben Sie sodann eine weitere Methode „ESA2Out“, mit der dieses Array aus der Datei ausgelesen und auf Konsole ausgegeben werden kann. Die Konsolenausgabe soll in sieben Zeilen mit jeweils elf Zeichen gegliedert werden – dies entspricht auch der vorstehenden Anordnung der Array-Elemente. Beachten Sie aber, dass eine Zeile im Array aus 33 Zeichen besteht – Ihre Konsolenausgabe soll hingegen elf Zeichen umfassen! Tipp: Wofür stehen die Werte im Array? Für diese gegliederte Ausgabe müssen Sie eine passende Steuerstruktur entwerfen.  Geben Sie den Quellcode an und zeigen Sie in einem Screenshot die Ausgabe der Methode „ESA2Out“.  Bei der eingereichten Variante wurden alle bytes mit einem Komma separiert geschrieben und nach jeweils 11 bytes wurde eine neue Zeile angefangen. Da hierbei alles als string abgespeichert wurde, mussten hier beim Einlesen und der Ausgabe einige Zwischenschritte gemacht werden. Zuerst wird die Datei zeilenweise gelesen (bis ein catch gefangen wird). Da wir als Ergebnis des Readers die Zeile als String erhalten, wird dieser über die Split-Methode in einen Array of Strings separiert (der Linq Befehl Where wird dabei genutzt, um das letzte leere Element wegzulassen). Über den Linq Befehl Select werden alle Elemente vom string in einen Byte konvertiert. Anschließend wird die "Sammlung bzw. IEnumerable Gruppe" in einen Array konvertiert. Dieser Byte Array wird über die ASCII Endocing klasse in den ASCII Buchstaben übersetzt und man erhält am Ende eine Tolle Konsolenausgabe.  namespace ESA\_2 {  class Program  {  public void ESA2In(string pfad,byte[] array)  {    StreamWriter write = new StreamWriter(File.Open(pfad,FileMode.Create));  for (int idx = 0; idx < array.Length; idx++)  {  if (idx > 0 && idx % 11 == 0)  {  write.WriteLine();  write.Write(array[idx].ToString());  write.Write(",");  }  else  {  write.Write(array[idx].ToString());  write.Write(",");  }  }  write.Close();  }  public void ESAOut(string path)  {  StreamReader read = new StreamReader(File.Open(path, FileMode.Open));  bool Continue = true;    while (Continue){  try  {  string string1 = read.ReadLine();  byte[] Zeichen\_AsASCII = string1.Split(',').Where(x => !x.Equals("")).Select(x => Convert.ToByte(x)).ToArray();  Console.WriteLine(System.Text.Encoding.ASCII.GetString(Zeichen\_AsASCII));      }  catch (NullReferenceException e)  {  Continue = false;  }  }    read.Close();  }  static void Main(string[] args)  {  Program test = new Program();    string pfad = @".\ESA2.txt";  byte[] array = {32, 32, 67, 67, 32, 32, 32, 35, 32, 35, 32,  32, 67, 32, 32, 67, 32, 32, 35, 32, 35, 32,  67, 32, 32, 32, 32, 32, 35, 35, 35, 35, 35,  67, 32, 32, 32, 32, 32, 32, 35, 32, 35, 32,  67, 32, 32, 32, 32, 32, 35, 35, 35, 35, 35,  32, 67, 32, 32, 67, 32, 32, 35, 32, 35, 32,  32, 32, 67, 67, 32, 32, 32, 35, 32, 35, 32 };  test.ESA2In(pfad, array);  test.ESAOut(pfad);  Console.ReadKey();  }  } }  Sie haben die Aufgabe sehr umständlich gelöst. Das Array hätten Sie mit einer Überladungsvariante der Write-Methode mit einer Zeile Code in die Datei schreiben können. Bei der Ausgabe hätten Sie dann nur eine Typkonvertierung nach “char” vornehmen müssen, um Zeichen auszugeben.  Punkte: 20/25 |  |
| 3. | Erstellen Sie für diese Einsendeaufgabe ein neues Projekt „ESA\_Projekt“, in das Sie den letzten Stand des Fliegerprojekts kopieren. Ergänzen Sie das Pro- jekt durch eine Programmierung, mit der die Positionsdaten eines Flugs in ei- ner Datei protokolliert werden können.  – Die Protokollierungsdatei soll in den Ausführordner des Programms („Release“ oder „Debug“) geschrieben werden  – Der Name jeder Protokolldatei soll nach dem Muster  <kennung>\_<Tag>-<Stunde>-<Minute>-<Sekunde>.bin  zusammengesetzt werden. Das lässt sich programmgesteuert erzeugen und unterscheidet mit dieser zeitabhängigen Codierung zuverlässig mehrere Protokolldateien.  – In die Datei soll zunächst ein String geschrieben werden, der folgende Informationen zusammenfasst:  Flug "<kennung>" (Typ "<typ>") startet an Position  "<x>-<y>-<h>" mit Zielposition "<x>-<y>-<h>".  Das ist der Header der Protokollierungsdatei, auf den sodann die Daten folgen. Auch hier sind wieder alle Ausdrücke mit spitzen Klammern durch konkrete Werte aus der Programmierung zu ersetzen.  – In die Datei sollen sodann fortlaufend die x-, y- und h-Daten der aktuellen Position *als int-Werte* geschrieben werden, bis der Flug beendet ist. Das sind sozusagen die Daten der Datei.  – Die Protokollierung soll vom Wert eines „Schalters“ abhängig gemacht werden, den Sie in der Klasse „Program“ als static-Variable definieren:  public static bool protokollieren = true;  Wenn dieser Schalter auf „true“ gesetzt ist, wird protokolliert, wenn er auf „false“ gesetzt ist, wird nicht protokolliert.  Bei dieser Aufgabe geht es nicht nur um die technische Realisierung vorgenannter Anforderungen, sondern vor allem um grundlegende Überlegungen objektorientierter Programmierung. Sie sollten sich z. B., wenn Sie den nötigen Writer deklarieren, klarmachen, dass ein Writer einem Flugschreiber entspricht und prüfen, welcher Fliegertyp überhaupt einen Flugschreiber nutzen kann. Beispiele für derartige Überlegungen objekt- orientierter Planung gab es in diesem Heft, aber auch schon bei der Planung des Projekts im Lernheft CSH01 – z. B. zur Ansiedlung des Transponders. Auf diesen Überlegungen aufbauend sind dann folgende Fragen zu beantworten: |  |
| a) Wo deklarieren Sie das Writer-Objekt, damit es an den Stellen im Programm, an denen es benötigt wird, auch bekannt ist?  Das Writer Objekt repräsentiert einen Flugschreiber und wird daher als Variable in einer unseren Klassen deklariert. Er ist also ein Member in unseren Klassen. Um die passende Klasse zu finden, habe ich folgende Überlegung gehabt : "Da der Writer den Typ benötigt und Typ nur im Düsenflugzeug deklariert wird, wird der Writer auch im Düsenflugzeug deklariert. Somit kann der Writer in jeder Methode des Düsenflugzeugs aufgerufen werden." Der Schreiber wird dabei als LocalWriter private deklariert und eine Eigenschaft Flugschreiber greift darauf zu (allerdings habe ich während der Implementierung doch lieber direkt auf die private Variable zugegriffen, als über die Eigenschaft).  class Düsenflugzeug : Starrflügelflugzeug  {  public Airbus typ;    private int sitzplaetze;  private int fluggaeste;  private BinaryWriter LocalWriter;  private string LocalWriter\_Path;  string datetime;  //public int Fluggaeste  //{  // set  // {  // if (sitzplaetze < (fluggaeste + value))  // Console.WriteLine("Keine Buchung:Die Fluggastzahl wuerde mit der Zubuchung von {0} Plaetzen die verfuegbaren Plaetze von {1} um {2} uebersteigen!", value, sitzplaetze, value + fluggaeste - sitzplaetze);  // else  // fluggaeste += value;  // }  // get  // { return fluggaeste; }  //}  public BinaryWriter FlugSchreiber  {  get  {  return this.LocalWriter;  }  set  {  // nicht wechselbar   }  }  Korrekt gelöst! |  |
| b) An welcher Stelle im Programm bauen Sie den Dateipfad zusammen, erstellen den Writer und schreiben den Header?  Der Programmpfad wird (in meinem Code) im Konstruktor des Düsenflugzeugs gebildet und dem Writer, der auch konstruiert wird, übergeben. Außerdem wird auch das Datum hier zusammengebaut (als String). Der Pfad und der der Writer könnten aber auch in der Startemethode gebaut (konstruiert) werden. Dann würde die Starte Methode aber noch einen weiteren Parameter benötigen (Programklasse müsste protokolieren an Starte übergeben). Der Header wird anschließend in der Startemethode gerufen (Logik : ein Flugzeug kann einen Schreiber haben, aber der Schreiber sollte erst gestartet werden, wenn das Flugzeug auch startet).  Wichtig : Damit ProgramTakten Düsenflugzeuge und Stahlflügelflugzeuge gleichermaßen aufrufen kann, habe ich die Methoden Starte, Transpond und Steuern als virtual deklariert und im Düsenflugzeug mit overwrite erweitert. Düsenflugzeug und Stahlflügelflugzeug haben nun beide diese 3 Methoden, aber Düsenflugzeug ruft neben den Methoden von Stahlflügelflugzeit seine weiteren "Erweiterungen" noch auf.  Pfad zuweisen :  public Düsenflugzeug(string kennung, Position pos, Airbus typ, Boolean SchreiberAn = true) : base(kennung, pos)  {  this.typ = typ;  this.pos = pos;  sitzplaetze = (int)typ;  this.datetime = DateTime.Now.Day.ToString() + "-" + DateTime.Now.Hour.ToString() + "-" + DateTime.Now.Minute.ToString() + "-" + DateTime.Now.Second.ToString();  if (SchreiberAn)  {  this.LocalWriter\_Path = @".\" + kennung.ToString() + "-" + datetime + ".bin";  this.LocalWriter = new BinaryWriter(File.Open(this.LocalWriter\_Path, FileMode.Create)); //FlugSchreiber(kennung, typ, FileLocation);  //LocalWriter.Close();  //Console.WriteLine("Flugschreiber von " + this.typ.ToString() + " mit der Kennung " + this.kennung.ToString() + " ist angeschaltet. ");  //}  //else  //{  // Console.WriteLine("Flugschreiber von " + this.typ.ToString() + " mit der Kennung " + this.kennung.ToString() + " ist ausgeschaltet.");  }  //Console.WriteLine("Der Flieger vom Typ {0} hat {1} Plaetze", typ, sitzplaetze);  }  Schreibe Header :  public override void Starte(Position zielpos, int streckeProTakt, int flughöhe, int steighöheProTakt, int sinkhöheProTakt)  {  base.Starte(zielpos, streckeProTakt, flughöhe, steighöheProTakt, sinkhöheProTakt);  if (LocalWriter != null)  {  LocalWriter.Write("Flug" + '"' + kennung + '"' + "(Typ " + typ.ToString() + ")" + "startet an position " + pos.x.ToString() + "-" + pos.y.ToString() + "-" + pos.h.ToString() + "mit Zielposition " + zielpos.x.ToString() + "-" + zielpos.y.ToString() + "-" + zielpos.h.ToString() + ".");    }    }  Korrekt gelöst! |  |
| c) An welcher Stelle im Programm schreiben Sie die Daten der x-, y- und h-Werte?  In der Transpond Methode wird die Write Methode aufgerufen, da hier sowieso die Position übertragen wird. Write kann einfach gerufen werden, da der Writer noch nicht geschlossen wurde.  public override void Transpond(string kennung, Position pos)  {  base.Transpond(kennung, pos);    if (LocalWriter != null)  {  this.LocalWriter.Write(pos.x.ToString() + " " + pos.y.ToString() + " " + pos.h.ToString());  }  }  Die Positionsdaten sollten als int-Werte in die Datei geschrieben werden. Der Ort zum Schreiben der Daten ist mit der Transpond-Methode korrekt gewählt. |  |
| d) An welcher Stelle im Programm schließen Sie den Writer?  Die Steuermethode meldet Transpond ab, sobald die Variable gelandet true ist, somit kann der Writer in Transpond nicht geschlossen werden. Somit ist Steuern die Methode, die den Status gelandet = true als erstes bemerkt (abruft) und geeignet, den Writer zu schließen.  public override void Steuern()  {  base.Steuern();  if (this.gelandet)  {  if (LocalWriter != null)  {  LocalWriter.Close();  }  }  }  Korrekt gelöst! |  |
| e) Wie stellen Sie jeweils sicher, dass diese Operationen nur dann erfolgen, wenn der „Schalter“ in der Programmklasse auf „true“ gestellt ist?  Im Konstruktor der Düsenflugzeugklasse wird als neuer Parameter protokolieren abgefragt. Falls sie true ist, wird ein neuer Writer konstruiert und der WriterVariable zugewiesen, wenn sie false ist, ist die WriterVariable des Düsenflugzeugs null. Starte, Transpond und Steuern prüfen immer, ob ein Writer vorhanden (nicht null) ist oder nicht vorhanden ist. Falls vorhanden, sollen Write bzw. Close gerufen werden. Eine Alternative wäre, eine Flugschreiberklasse als Kindklasse des BinarayWriters zu deklarieren und eine bool Variable „TurnOn“ zu deklarieren. Der Flugschreiber könnte dann immer konstruiert werden, aber seine Variable „TurnOn“ kann true or false sein. In Starte, Transpond und Steuern kann nun die Variable vom (immer vorhandenen) Flugschreiber geprüft werden.  (siehe alle MEthoden vorher – sie fragen immer, ob Schreiber vorhanden ist) |  |
| Geben Sie jeweils den Programmcode an und erläutern Sie kurz Ihre Programmierung. |  |
| **Hinweis**:  BinaryWriter kennt eine Methode  public virtual void Write ( string value )  … die nicht nur den als Parameter anzugebenden String in die Datei schreibt, sondern davor eine Information über die Länge des Strings speichert (vergleichen Sie die .NET-Dokumentation).  BinaryReader kennt eine spiegelbildliche Methode  public virtual strng ReadString ()  … die zunächst die in der Binärdatei gespeicherte Längeninformation auswer- tet und sodann präzise die Länge des nachfolgenden String liest, sodass der Lesezeiger für weitere Leseoperationen genau auf das nachfolgende Byte gesetzt wird.  Punkte: 40/45 |  |
| 4. | Ergänzen Sie in der Programmklasse „FliegerProjekt“ (zu Einsendeaufgabe 3) eine Methode  public void ESA4Out(string protokollpfad)  Diese Methode soll die Binärdatei eines Fluges, die mit der Programmierung in ESA 3 erstellt wird, zurücklesen und deren Inhalt auf Konsole ausgeben. Im Ergebnis könnte das für zwei Flüge wie in Abb. D.1 aussehen:  479  ***Abb. D.1:*** *Ausschnitt zur Ausgabe der Flugprotokollierung in eine Datei für zwei Flüge auf Grundlage der init-Dateien zu „LH 3000“ und „LH 500“ über die Methode „ESA4Out“*  Aufrufe dieser Methode „ESA4Out“ ergänzen Sie am Ende der Programmsteuerung in „ProgrammTakten“.  Damit die Methode „ESA4Out“ immer den zuletzt protokollierten Flug ausgeben kann, speichern Sie den Pfad auf dessen Protokollierungsdatei in einer neuen Property, die für den Abruf lediglich mit einem get-Accessor ausgestattet wird. Diese Property wird dort angelegt, wo auch die Protokollierung erfolgt.  Bei der Programmierung von „ESA4Out“ ist die innere Dateistruktur der Protokolldatei zu beachten, so dass in der Konsole zunächst der Header und sodann die Flugdaten ausgegeben werden. Die Flugdaten beginnen mit der Start- und enden mit der erreichten Zielposition. Wegen der groben Taktung und der daraus folgenden groben Positionsberechnungen werden die vorge-gebene und die real erreichte Zielposition voneinander abweichen.  Erläutern Sie die wesentliche Schritte Ihrer Lösung mit Codeauszügen.  …  Die Methode bekommt lediglich den Speicherpfad als Parameter und (falls die Protokollieren Variabel true ist) wird ein BinaryReader gerufen (mit open flag), Es wird nun die erste Zeile gelesen und ausgegeben.  Dabei wird die PadRight Methode genutzt, um den String richtig einzurücken. Der Reader wird so oft gerufen, bis das Ende des Files erreicht ist. Dies wird über while (Variabel) try catch erreicht. Solange das Ende nicht erreicht wurde, versuch die Zeile zu lesen und bei Erfolg gib das ganze über die Konsole aus (mitteils der PAdMethode zum einrücken).  Die Methode wird dabei so aufgerufen, dass der Übergabeparameter der Pfad Eigenschaft des Düsenflugzeugs entspricht (wie in der Aufgabe verlangt – Schreibervariabel ist eine Variabel vom Düsenflugzeug, also sollte die Pfadvariabel es auch sein).  Aufruf der Methode im Programm :  public static void ProgrammTakten()  {    Düsenflugzeug neuflieger = new Düsenflugzeug("LH5000", new Position(3500, 1500, 180),Airbus.A300, Program.protokollieren);  neuflieger.Starte(new Position(1000, 500, 200), 200,300,20,10);  Program.fliegerRegister += neuflieger.Steuern;  Starrflügelflugzeug neuflieger1 = new Starrflügelflugzeug("LH900", new Position(3000, 2000, 100));  neuflieger1.Starte(new Position(1000, 500, 200), 260, 350,25,15);  Program.fliegerRegister += neuflieger1.Steuern;  while (Program.fliegerRegister != null)   {  Program.fliegerRegister();    Thread.Sleep(1000);  }  Program newprogramm = new Program();  newprogramm.ESA4Out(neuflieger.Pfad);  }  Methode an sich :  public void ESA4Out(string protokollpfad)  {  if (protokollieren == true)  {  this.LokalReader = new BinaryReader(File.Open(protokollpfad, FileMode.Open));  var Puffer = LokalReader.ReadString();  Console.Write((Puffer + "\n").PadRight(10));  Console.Write("\n");  bool goon = true;  while (goon)  {  try  {  Puffer = LokalReader.ReadString();  var Positions = Puffer.Split(' ');    Console.Write("".PadRight(5));  Console.Write(Positions[0].PadRight(10));  Console.Write(Positions[2].PadRight(10));  Console.Write(Positions[4].PadRight(10) + "\n");  }  catch (EndOfStreamException e)  {  goon = false;  }  }  LokalReader.Close();  }  else  {  //Console.WriteLine("Ich bin fertig!");  }  }  Diese Lösung ist natürlich nicht die Lösung, die Sie programmieren müssten, wenn Sie Strings und int-Werte in der vorherigen Aufgabe in die Datei geschrieben hätten. Dann wäre die Lösung etwas aufwendiger.  Punkte: 12/15 |  |
|  | Reichen Sie mit Ihren Ausführungen zu den Einsendeaufgaben auch die vollständigen Quellcodes der beiden Projekte zu Einsendeaufgabe 2 bzw. den Einsendeaufgaben 3 (im Verbund mit 4) ein. |  |

Erreichte Punktzahl: 82/100

Note: gut (2)

Sonntag, 7. Mai 2017

Hallo Frau Zang,

an manchen Stellen müssen Sie Ihre Lösungen noch verbessern. Insgesamt eine gute Arbeit.

Mit freundlichen Grüßen

Jochen Senkbeil