МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра стратегічного управління

Індивідуальне домашнє завдання

з дисципліни «Кросплатформене програмування»

Виконав:

ст. гр. КН-27

Перевірив:

доц. каф. САИТ

Марченко І. І.

Харків 2019

Product.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  namespace ConsoleApp1  {  [Serializable]  public class Product : IName<Product>, IPrice<Product>  {  public Decimal \_price;  public String name { get; set; }  public OnPriceChanged Editing;  public Decimal price  {  get  {  return \_price;  }  set  {  try  {  if (value < 0)  {  throw (new MyException("The price must be greater or equal to zero (set to 0)"));  }  if (Editing != null) Editing(price, value);  \_price = value;  }  catch (MyException e)  {  //Console.WriteLine("Smth went wrong: {0}", e.Message);  }  }  }    public int CompareTo(Product other)  {  return name.CompareTo(other.name);  }  public override String ToString() { return "name " + name + " price " + price; }  }  } |

Book.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  [Serializable]  public class Book : Product//, IPrice<Book>  {  public String isbn { get; set; } // International Standard Book Number  public String genre { get; set; }  public List<String> authors { get; set; }  public Book(String name, Decimal price, String isbn, String genre, List<String> authors)  {  this.name = name;  this.price = price;  this.isbn = isbn;  this.genre = genre;  this.authors = authors;  }  public Book() { }  public String get\_author(int indx)  {  return authors[indx];  }  public override string ToString()  {  return $"NAME: {this.name}, PRICE: {this.price}\n" +  $"ISBN: {this.isbn}, GENRE: {this.genre}, Authors {String.Join(", ", this.authors.ToArray())}";  }  }  } |

Journal.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  [Serializable]  class Journal : Product  {  public String issn { get; set; } // International Standard Serial Number  public String type { get; set; } = "journal"; // Either paper or journal or anything else with ISSN in it  public Journal() { }  public Journal(String type, String name, Decimal price, String issn)  {  this.type = type;  this.name = name;  this.price = price;  this.issn = issn;  }  public override string ToString()  {  return $"NAME: {this.name}, PRICE: {this.price}\n" +  $"ISSN: {this.issn}, TYPE: {this.type}";  }  }  } |

Container.cs

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Runtime.Serialization;  using System.Runtime.Serialization.Formatters.Binary;  using System.IO;  namespace ConsoleApp1  {  //[Serializable]  public class Container<T> : IEnumerable<T>//, ISerializable  where T : Product  {  public decimal Sum\_total { get; set; }  private T[] Product\_array { get; set; }  public Container()  {  }  public Container<T> SortByName()  {  if (Product\_array != null)  for (int i = 1; i < Product\_array.Length; ++i)  {  T pr = Product\_array[i];  int j = i - 1;  while (j >= 0 && Product\_array[j].CompareTo(pr) == 1)  {  Product\_array[j + 1] = Product\_array[j];  j = j - 1;  }  Product\_array[j + 1] = pr;  }  return this;  }  public Container<T> SortByPrice()  {  if (Product\_array != null)  for (int i = 1; i < Product\_array.Length; ++i)  {  T pr = Product\_array[i];  int j = i - 1;  while (j >= 0 && Product\_array[j].Price > pr.Price)  {  Product\_array[j + 1] = Product\_array[j];  j = j - 1;  }  Product\_array[j + 1] = pr;  }  return this;  }  public void Add(T product)  {  if (Product\_array is null)  {  Product\_array = new T[] {};  }  T[] temp = new T[Product\_array.Length + 1]; // Creating new temporary array  for (int i = 0; i < Product\_array.Length; i++)  {  temp[i] = Product\_array[i]; // Copying elements  }  product.Editing += priceChanged;  temp[Product\_array.Length] = product;  Sum\_total += product.Price;  Product\_array = temp; // Replacing array    }  private void priceChanged(decimal old\_price, decimal new\_price)  {  Sum\_total += -old\_price+new\_price;  }  public T this[int i]  {  get { return Product\_array[i]; }  set  {  Sum\_total += -Product\_array[i].Price + value.Price;  Product\_array[i] = value;  }  }  public T this[String str]  {  get  {  int count = 0;  T pr = default(T);  foreach (T p in Product\_array.Where(p => p is IName<T>))  {  if (p.Name == str)  {  count++;  pr = p;  }  }  if (count == 1) return pr;  return default(T);  }  set  {  int count = 0;  T pr = default(T);  foreach (T p in Product\_array)  {  if (p.Name == str)  {  count++;  pr = p;  }  }  if (count == 1)  {  Sum\_total = -pr.Price + value.Price;  pr = value;  }  else if (count == 0) throw new MyException($"Zero instances of objects with name {str}");  else throw new MyException($"There are multiple instances of objects within container with name {str}");  }  }  public void deleteByIndex(int i)  {  if (i < 0 || i > Product\_array.Length) throw (new MyException("Nonexistent index: " + i));  Product\_array[i].Editing -= priceChanged;  T[] temp = new T[Product\_array.Length - 1];  Sum\_total -= Product\_array[i].Price;  for (int j = 0; j < Product\_array.Length-1; j++)  {  if (j < i) temp[j] = Product\_array[j];  else temp[j] = Product\_array[j + 1];  }  Product\_array = temp;  }  //public override string ToString()  //{  // StringBuilder sb = new StringBuilder();  // foreach (T product in Product\_array)  // {  // sb.Append(product.ToString() + "\n===\n");  // }  // return sb.ToString();  //}  public Container<T> Sort(Func<T, T, int> func)  {  // Insertion Sort  for (int i = 1; i < Product\_array.Length; ++i)  {  T pr = Product\_array[i];  int j = i - 1;  while (j >= 0 && func(Product\_array[j], pr) == 1)  {  Product\_array[j + 1] = Product\_array[j];  j = j - 1;  }  Product\_array[j + 1] = pr;  }  return this;  }  public IEnumerable GetReversed()  {  for (int i = Product\_array.Length - 1; i >= 0; i--)  {  yield return Product\_array[i];  }  }  public IEnumerable GetWithSubstring(String str)  {  foreach (T t in Product\_array.Where(t => t is IName<T>))  if (t.Name.ToLower().Contains(str.ToLower()))  yield return t;  }  internal IEnumerable<T> GetOrdered(Func<T, T, int> p)  {  T[] new\_array = new T[Product\_array.Length];  Array.Copy(Product\_array, new\_array, Product\_array.Length);  Sort(p);  foreach (T pr in Product\_array)  {  yield return pr;  }  Product\_array = new\_array;  }  internal List<T> FindAll(Func<T, bool> p)  {  List<T> list = new List<T>();  foreach (T pr in Product\_array)  {  if (p(pr)) list.Add(pr);  }  return list;  }  internal T Find(Func<T, bool> p)  {  foreach (T pr in Product\_array)  {  if (p(pr)) return pr;  }  return default(T);  }  IEnumerator<T> IEnumerable<T>.GetEnumerator()  {  return (IEnumerator<T>)GetEnumerator();  }  IEnumerator IEnumerable.GetEnumerator()  {  return (IEnumerator)GetEnumerator();  }  public ContainerEnum<T> GetEnumerator()  {  return new ContainerEnum<T>(Product\_array);  }  public override string ToString()  {  var arr = from pr in this  group pr by pr.GetType() into g  select new { type = g.Key.Name, price = g.Average(val => val.Price), objects = from p in g select p };  String txt = "";  foreach (var gr in arr)  {  txt += gr.type.ToUpper() + "\r\n";  foreach (var o in gr.objects)  {  txt += $"- {o}\r\n";  }  }  if (Product\_array != null)  {  txt += "\r\n\r\n";  foreach (var pr in arr)  {  txt += $"Type: {pr.type}, AvgPrice: {pr.price}\r\n";  }  }  return txt;  }  public void WriteToTxt(String name)  {  StreamWriter sw = new StreamWriter($"C:\\Users\\Richer\\OneDrive\\Studying\\Прогач\\IDZ\\ConsoleApp1\\ConsoleApp1\\{ name }.txt");  sw.Write("Total sum: " + Sum\_total + "\r\n" + this);  sw.Close();  }  }  } |

ContainerEnum.cs

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  public class ContainerEnum<T> : IEnumerator<T> // Standart iterator  {  public T[] product\_array;  int position = -1;    public ContainerEnum(T[] product\_array)  {  this.product\_array = product\_array;  }  public bool MoveNext()  {  position++;  return (position < product\_array.Length);  }  public void Reset()  {  position = -1;  }  public void Dispose()  {  //throw new NotImplementedException();  }  T IEnumerator<T>.Current  {  get  {  try  {  return product\_array[position];  }  catch (IndexOutOfRangeException)  {  throw new InvalidOperationException();  }  }  }  public object Current  {  get  {  try  {  return product\_array[position];  }  catch (IndexOutOfRangeException)  {  throw new InvalidOperationException();  }  }  }  }  } |

Delegates.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  public delegate void OnPriceChanged(decimal old\_price, decimal new\_price);  } |

IName<T>.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  public interface IName<T> : IComparable<T>  {  String name { get; set; }  }  } |

IPrice<T>.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  public interface IPrice<T>  {  decimal price { get; set; }  }  } |

MyException.cs

|  |
| --- |
| using System;  namespace ConsoleApp1  {  public class MyException : Exception  {  public MyException(string message) : base(message)  {    }  public MyException()  {  }  public MyException(string message, Exception inner)  : base(message, inner)  {  }  }  } |

Program.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Runtime.Serialization.Formatters.Binary;  using System.Xml.Serialization;  using System.Reflection;  using System;  using System.Xml;  using System.Collections;  namespace ConsoleApp1  {  class Program  {  static void Main(string[] args)  {  Container<Product> container1 = new Container<Product>();  // Adding to container  Console.WriteLine("Adding to container...");  container1.Add(new Book("The Great Gatsby", -5, "978-3-16-148410-0", "Fiction", new List<String>() { "F. Scott Fitzgerald" }));  container1.Add(new Journal("Journal", "National Geographic", 5, "2049-3630"));  container1.Add(new Book("Gone With The Wind", 40, "439-4-12-14321-8", "Novel", new List<String>() { "Margaret Mitchell" }));  container1.Add(new Journal("Paper", "Time", 15, "2021-5431"));  container1.Add(new Book("Mud, Sweat and Tears", 40, "433-4-12-12345-8", "Biography", new List<String>() { "Bear Grylls" }));  container1.Add(new Book("The Art of the Deal", 30, "111-4-12-22222-8", "Biography", new List<String>() { "Donal Trump" }));  container1.Add(new Book("The Catcher in the Rye", 22, "543-4-12-14321-8", "Novel", new List<String>() { "J. D. Salinger" }));  container1.Add(new Book("The Jungle Book", 12, "555-4-12-14321-8", "Children's Book", new List<String>() { "Rudyard Kipling" }));  ////================================================  //// Removing from container  //Console.WriteLine("Removing from container...");  //container1.deleteByIndex(1);  //container1.deleteByIndex(111);  ////================================================  //// Getting info in the string format  //Console.WriteLine("Getting info in the string format...");  //Console.WriteLine(container1);  ////================================================  //// Getting an element by index or name  //Console.WriteLine("Getting an element by index or name...");  //Console.WriteLine(container1[3]);  //Console.WriteLine(container1["The Catcher in the Rye"]);  //Console.WriteLine(container1["The Catcher in the Ry"]);  ////================================================  //// Getting ordered data  //Console.WriteLine("SORTED BY NAME\n" + container1.SortByName());  //Console.WriteLine("SORTED BY PRICE\n" + container1.SortByPrice());  ////================================================  //// Iterating  //Console.WriteLine("Iterating through container...");  //foreach (Product pr in container1)  //{  // Console.WriteLine(pr);  //}  //Console.WriteLine();  //Console.WriteLine("Iterating through reversed container: ");  //foreach (Product pr in container1.GetReversed())  //{  // Console.WriteLine(pr);  //}  //Console.WriteLine();  //Console.WriteLine("Iterating through container (with elements containing substring): ");  //foreach (Product pr in container1.GetWithSubstring("ional"))  //{  // Console.WriteLine(pr);  //}  //Console.WriteLine();  //Console.WriteLine("Iterating through ordered container: ");  //foreach (Product pr in container1.GetOrdered((Product pr1, Product pr2) => pr1.price.CompareTo(pr2.price)))  //{  // Console.WriteLine(pr);  //}  ////================================================  //// Saving to txt  //container1.writeToTxt("container");  ////================================================  //// Serialization and Deserialization  //Stream stream = File.Open("ContainerData.dat", FileMode.Create);  //BinaryFormatter bf = new BinaryFormatter();  //bf.Serialize(stream, container1);  //stream.Close();  //container1 = null;  //stream = File.Open("ContainerData.dat", FileMode.Open);  //bf = new BinaryFormatter();  //container1 = (Container<Product>)bf.Deserialize(stream);  //stream.Close();  ////================================================  //// Getting sorted data  //Console.WriteLine("Getting sorted data...");  //Console.WriteLine("SORTED BY NAME\n" + container1.Sort((Product pr1, Product pr2) => String.Compare(pr1.name, pr2.name)));  //Console.WriteLine("SORTED BY PRICE\n" + container1.Sort((Product pr1, Product pr2) => pr1.price.CompareTo(pr2.price)));  ////================================================  //// Finding elements  //Console.WriteLine("Finding elements...");  //Console.WriteLine();  //Console.WriteLine("FindAll: ");  //foreach (Product pr in container1.FindAll((Product pr) => pr.name.ToLower().Contains("the".ToLower())))  //{  // Console.WriteLine(pr);  //}  //Console.WriteLine();  //Console.WriteLine("Find: ");  //Console.WriteLine(container1.Find((Product pr) => pr.name.ToLower().Contains("the".ToLower())));  ////================================================  ////Getting total sum in container  ////Console.WriteLine("Getting total sum in container...");  //Console.WriteLine(container1.sum\_total);  //container1[0].price += 5;  //Console.WriteLine(container1.sum\_total);  //container1.deleteByIndex(3);  //Console.WriteLine(container1.sum\_total);  ////================================================  //// LINQ  //Console.WriteLine("LINQ...");  //Console.WriteLine("Min price: " + (from el in container1 where (el.Price == container1.Min(elem => elem.Price)) select el).First());  //Console.WriteLine("Max price: " + (from el in container1 where (el.Price == container1.Max(elem => elem.Price)) select el).First());  //Console.WriteLine("Min price: " + container1.Min(elem => elem.Price));  //Console.WriteLine("Max price: " + container1.Max(elem => elem.Price));  //Console.WriteLine(container1);  Book b = new Book("The Great Gatsby", 10, "978-3-16-148410-0", "Fiction", new List<String>() { "F. Scott Fitzgerald", "Mark Twaine" });  Journal j = new Journal("Journal", "National Geographic", 5, "2049-3630");    int[] arr = new int[] { 1, 2, 3 };  double[][] arr2 = new double[][] { new double[] { 1, 2 }, new double[] { 3, 4 } };  List<int> list = new List<int>() { 9, 2, 3 };  Product[] arr3 = new Product[] { b, j };  Stream stream = new MemoryStream();  List<List<int[]>> list2 = new List<List<int[]>>() {  new List<int[]>() { new int[] { 1, 2 }, new int[] { 1, 2 } },  new List<int[]>() { new int[] { 1, 2 }, new int[] { 3, 4 }, new int[] { 5, 6 } },  new List<int[]>() { new int[] { 1, 2 }, new int[] { 7, 8 } },  };  List<List<int>> list3 = new List<List<int>>() { new List<int>() { 1, 2 }, new List<int>() { 3, 4 } };  object[] arr4 = new object[] { new int[] { 1 }, "asd" };  Console.WriteLine("SERIALIZATION:\n");  Serializator.Serialize(list3, stream);  stream.Seek(0, SeekOrigin.Begin);    Console.WriteLine("\nDESERIALIZATION:\n");  Console.WriteLine(Serializator.Deserialize(stream));  //Container<Product> cont2 = (dynamic)Serializator.Deserialize(stream);  //Console.WriteLine(cont2);  Console.ReadKey();  }  }  } |

Serializator.cs

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  public static class Serializator  {  private static void getValue(Type propType, Stream stream, ref object result, String propName = null)  {  BinaryReader br = new BinaryReader(stream);  //if (Type.GetTypeCode(propType) == TypeCode.String)  //{  // String stringValue = br.ReadString();  // //Console.WriteLine(stringValue);  // if (propName == null)  // result = stringValue;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { stringValue });  //}  if (propType.IsValueType || Type.GetTypeCode(propType) == TypeCode.String)  {  object o = br.GetType().GetMethod($"Read{propType.Name}").Invoke(br, new object[] { });  if (propName == null)  result = o;  else  result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { o });  }  //switch (propType.Name)  //{  // case "Boolean":  // bool BoolValue = br.ReadBoolean();  // //Console.WriteLine(BoolValue);  // if (propName == null)  // result = BoolValue;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { BoolValue });  // break;  // case "Char":  // int CharValue = br.ReadChar();  // //Console.WriteLine(CharValue);  // if (propName == null)  // result = CharValue;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { CharValue });  // break;  // case "Int32":  // int Int32Value = br.ReadInt32();  // //Console.WriteLine(Int32Value);  // if (propName == null)  // result = Int32Value;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { Int32Value });  // break;  // case "Int16":  // int Int16Value = br.ReadInt16();  // //Console.WriteLine(Int16Value);  // if (propName == null)  // result = Int16Value;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { Int16Value });  // break;  // case "Int64":  // long Int64Value = br.ReadInt64();  // //Console.WriteLine(Int64Value);  // if (propName == null)  // result = Int64Value;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { Int64Value });  // break;  // case "Double":  // double DoubleValue = br.ReadDouble();  // //Console.WriteLine(DoubleValue);  // if (propName == null)  // result = DoubleValue;  // else  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { DoubleValue });  // break;  // case "Decimal":  // Decimal DecimalValue = br.ReadDecimal();  // //Console.WriteLine(DecimalValue);  // result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetField | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { DecimalValue });  // break;  // default:  // break;  // }  //}  }  public static Object Deserialize(Stream stream)  {  BinaryReader br = new BinaryReader(stream);  String name = br.ReadString();  String innerType = "";  Type t = Type.GetType(name);  Object result = null;  if (Type.GetTypeCode(t) == TypeCode.String || t.IsValueType)  {  getValue(t, stream, ref result);  }  else  {  if (t.IsArray)  {  String storedType = br.ReadString();  int amount = br.ReadInt16(); // Getting the size of the array  //Object[] array = new object[amount];  Array array = (dynamic)Array.CreateInstance(Type.GetType(storedType),amount);  for (int j = 0; j < amount; j++)  {  array.SetValue(Deserialize(stream),j);  }  result = array;  //result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { array });  }  else  {  if (t.IsGenericType && t.GetGenericTypeDefinition() == typeof(List<>))  {  String storedType = br.ReadString();  int amount = br.ReadInt16(); // Getting the size of the array  var listType = typeof(List<>);  var generic = listType.MakeGenericType(Type.GetType(storedType));  Object list = Activator.CreateInstance(generic);  for (int j = 0; j < amount; j++)  {  object item = Deserialize(stream);  list.GetType().GetMethod("Add").Invoke(list, new object[] { item });  }  result = list;  }  else  {  result = Activator.CreateInstance(t);  bool isGeneric = br.ReadBoolean();  if (isGeneric)  {  innerType = br.ReadString(); // Getting inner type of container  }  int size = br.ReadInt16(); // Getting amount of properties in the object  Object[] args = new object[size];  for (int i = 0; i < size; i++)  {  String propName = br.ReadString(); // Getting name of a member  String type = br.ReadString(); // Getting the next stored type  Type propType = Type.GetType(type);  //Console.Write(propName + ": ");  getValue(propType, stream, ref result, propName);  String storedType;  int amount;  if (propType.IsGenericType)  {  storedType = br.ReadString();  var listType = typeof(List<>);  var generic = listType.MakeGenericType(Type.GetType(storedType));  Object list = Activator.CreateInstance(generic);  amount = br.ReadInt16(); // Getting the size of the list    for (int j = 0; j < amount; j++)  {  //String tempType = br.ReadString();  object item = Deserialize(stream);  list.GetType().GetMethod("Add").Invoke(list, new[] { item });  }  result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetProperty, Type.DefaultBinder, result, new Object[] { list });  }  if (propType.IsArray)  {  storedType = br.ReadString();  amount = br.ReadInt16(); // Getting the size of the array  Array array = Array.CreateInstance(Type.GetType(storedType), amount);  for (int j = 0; j < amount; j++)  {  array.SetValue(Deserialize(stream), j);  }  result.GetType().InvokeMember(propName, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.SetProperty, Type.DefaultBinder, result, new object[] { array });  }  }  }  }  }  return result;  }    public static void Serialize(object obj, Stream stream)  {  BinaryWriter Bw = new BinaryWriter(stream);  Bw.Write(obj.GetType().FullName); // Getting name of stored type  if (obj.GetType().IsValueType || Type.GetTypeCode(obj.GetType()) == TypeCode.String)  {  Bw.Write((dynamic)obj);  }  else  {  if (obj.GetType().IsArray) // If member is an array  {  Bw.Write(obj.GetType().GetElementType().FullName); // Writes name of type stored in the array to stream  Bw.Write((Int16)(obj as Array).Length);  foreach (object o in (obj as Array))  {  Serialize(o, stream);  }  }  else  {  if (obj.GetType().IsGenericType && obj.GetType().GetGenericTypeDefinition() == typeof(List<>)) // If field is a List  {  Bw.Write(obj.GetType().GetGenericArguments()[0].FullName); // Writing name of a type that's stored in a list  Bw.Write((Int16)(obj as ICollection).Count); // Writing the length of the collection to a stream  foreach (object o in (obj as IEnumerable))  {  Serialize(o, stream);  }  }  else  {  if (obj.GetType().IsGenericType)  {  Bw.Write(true);  Bw.Write(obj.GetType().GetGenericArguments()[0].FullName); // Storing the name of container's inner type  }  else  {  Bw.Write(false);  }  PropertyInfo[] properties = obj.GetType().GetProperties(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public).Where(val => val.Name != "Item").ToArray();    //MemberInfo[] members = obj.GetType().GetFields(Bf).Cast<MemberInfo>()  // .Concat(obj.GetType().GetProperties(Bf)).  // Where(val => !(typeof(MulticastDelegate).IsAssignableFrom(GetTypeForMember(val)))).  // Where(val => val.Name != "Item").  // ToArray();  Bw.Write((Int16)properties.Length); // Writing the amount of fields in the object  foreach (PropertyInfo property in properties)  {  Bw.Write(property.Name); // Storing name of a member  Bw.Write(property.PropertyType.FullName); // Storing the type's name in a stream  //Bw.Write(Type.GetTypeCode(member.PropertyType).);  //Console.WriteLine(property.Name + ": " + property.GetValue(obj));  if (property.PropertyType.IsValueType || Type.GetTypeCode(property.PropertyType) == TypeCode.String)  {  Bw.Write((dynamic)property.GetValue(obj));  }  if (property.PropertyType.IsGenericType && property.PropertyType.GetGenericTypeDefinition() == typeof(List<>)) // If field is a List  {  Bw.Write(property.PropertyType.GetGenericArguments()[0].FullName); // Writing name of a type that's stored in a list  Bw.Write((Int16)(property.GetValue(obj) as ICollection).Count); // Writing the length of the collection to a stream  foreach (object o in (property.GetValue(obj) as IEnumerable))  {  Serialize(o, stream);  }  }  if (property.GetValue(obj).GetType().IsArray) // If member is an array  {  Bw.Write(property.GetValue(obj).GetType().GetElementType().FullName); // Writes name of type stored in the array to stream  Bw.Write((Int16)(property.GetValue(obj) as Array).Length);  foreach (object o in (property.GetValue(obj) as Array))  {  Serialize(o, stream);  }  }  }  }  }  }  }  }  } |