Отчет по лабораторной работе №1

“Исследование отличий

между структурным и объектным подходом”

по дисциплине

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Выполнил студент гр. ИВТ/б-22о

Горбенко К.Н.

Проверил:

Тимофеев И.С.

1. **Цель работы:** исследование основных средств определения класса, создания объектов класса. Исследование отличий структурного и объектного подходов.
2. **Задание на работу**: написать две программы, которые, в соответствии с вариантом №8, выполняют следующие действия для заданных структур и классов:  
    Первая программа:

* ввод с клавиатуры данных в переменную структуры;
* вывод на дисплей введенных данных;
* обработка введенных данных способом, указанным в варианте и вывод на дисплей результатов обработки;

Для демонстрации работы функций создать две переменные заданного типа, вывести на экран меню, считать выбор пользователя, выполнить соответствующую функцию. Меню должно содержать пункты:

* чтение данных с клавиатуры;
* вывод данных на дисплей;
* обработка данных;
* выход.

Вторая программа должна быть написана на базе первой, вместо структуры в ней необходимо использовать класс, поля которого совпадают с полями структуры, а действия, выполняемые функциями в первой программе, должны выполнять методы класса.

Условие для **варианта № 8**:

Структура с именем TRAIN, поля:

* название пункта назначения;
* номер поезда;
* время отправления.

Способ обработки - определение результата сравнения: верно ли, что время отправления обоих поездов совпадает.

1. **Код программы на языке C#:** Структура Train:  
     
    //Train struct. Normally should be named just Train.  
    //Called like this to enable to add another   
    //implementation of same object within one solution  
    public struct TrainAsStruct : ITrain  
    {  
    public string Destination { get; set; }  
    public int Number { get; set; }  
    public DateTime Departure { get; set; }  
     
    //Struct should normally have a constructor.  
    //No constructor here because of the attempt to   
    //stick to condition of task  
     
    //Constructor gets created automatically  
     
    //Method to compare two ITrain objects  
    public override bool Equals(object obj)  
    {  
    var train = (ITrain)obj;  
     
    return Departure.Equals(train.Departure);  
    }  
     
    //Method to get full object status  
    public override string ToString()  
    {  
    //string interpolation used  
    return $"Train number {Number,5};" +  
    $"destination {Destination,20};" +  
    $"departure {Departure.ToString()};" +  
    $" type {GetType()}";  
    }  
    }

Класс Train:

|  |
| --- |
| //Train class. Normally should be named just Train  //Called like this to enable to add another   //implementation of same object within one solution  public class TrainAsClass : ITrain  {  public string Destination { get; set; }  public int Number { get; set; }  public DateTime Departure { get; set; }   //constructor  public TrainAsClass(string destination, int number, DateTime departure)  {  //null checks to prevent using of invalid  //parameters  if (String.IsNullOrWhiteSpace(destination)) throw new ArgumentNullException(nameof(destination));   Destination = destination;  Number = number;  Departure = departure;  }   //method to compare two ITrain objects  public override bool Equals(object obj)  {  var anotherTrain = obj as ITrain;  //same null check  if (anotherTrain == null) throw new InvalidCastException(nameof(obj));   return Departure.Equals(anotherTrain.Departure);  }   //method to get a full object status  public override string ToString()  {  //string interpolation used  return $"Train number {Number,5};" +  $"destination {Destination,20};" +  $"departure {Departure.ToString()};" +  $"type {GetType()}";  }  } |

Класс ITrain:

|  |
| --- |
| //Interface to enable examing objects of Train struct  //and class as different implementation of same logic  public interface ITrain  {  string Destination { get; set; }   int Number { get; set; }   DateTime Departure { get; set; }   bool Equals(object obj);   string ToString();  } |

Класс Program:

|  |
| --- |
| class Program  {  static void Main()  {  int userOption = -1;  var enteredTrains = new List<ITrain>();   do  {  //try-catch blocks to prevent program executing  //is extensive situation  try  {  Console.Clear();  Console.WriteLine("Enter your option:\n" +  "1 - Create new train\n" +  "2 - Write all trains details\n" +  "3 - Compare objects\n" +  "4 - Exit");   userOption = int.Parse(Console.ReadLine());   switch (userOption)  {  case 1:  enteredTrains.Add(CreateObject());  break;  case 2:  WriteTrainsToConsole(enteredTrains);  break;  case 3:  if (AreSelectedTrainsEqual(enteredTrains))  Console.WriteLine("Selected trains are equal");  else Console.WriteLine("Selected trains are not equal");  Console.ReadLine();  break;  default:  Console.WriteLine("You have entered a wrong option, try again");  continue;  }   }  catch (Exception e)  {  Console.WriteLine("Something went wrong. Try again");  Console.WriteLine(e.Message);  Console.ReadLine();  continue;  }  } while (userOption != 4);  }   private static ITrain CreateObject()  {  int userOption = -1;   //endless while loop to not to finish creating  //of an object when invalid option is entered   do  {  Console.Clear();  Console.WriteLine("What data type should new train be?\n" +  "1 - Structure\n" +  "2 - Class");   userOption = int.Parse(Console.ReadLine());   if (userOption == 1) return CreateTrainStructure();  if (userOption == 2) return CreateTrainClass();  } while (true);  }   //Method to create instance of train structure  //endless while loops for each input ask  //to make sure that appropriate value has been entered   private static TrainAsStruct CreateTrainStructure()  {  var train = new TrainAsStruct();   while (train.Destination == null)  {  Console.Clear();  Console.WriteLine("Enter train destination:");  var destination = Console.ReadLine();   train.Destination = destination;  }   while (train.Number == 0)  {  Console.Clear();  Console.WriteLine("Enter train number:");  var trainNumber = int.Parse(Console.ReadLine());   train.Number = trainNumber;  }   while (train.Departure == DateTime.MinValue)  {  //try-catch blocks to make sure that user have   //entered valid value. When value is invalid,   //user gets asked again  try  {  Console.Clear();  Console.WriteLine("Enter train departure time:");  train.Departure = DateTime.Parse(Console.ReadLine());  }  catch  {  Console.WriteLine("The input string was not in the correct format. Try again");  continue;  }  }  return train;  }   //Method to create instance of train class.  //Endless while loops for each input ask  //to make sure that appropriate value has been entered  private static TrainAsClass CreateTrainClass()  {  string destination = null;  var trainNumber = 0;  DateTime departure = DateTime.MinValue;   while (destination == null)  {  Console.Clear();  Console.WriteLine("Enter train destination:");  destination = Console.ReadLine();  }   while (trainNumber == 0)  {  Console.Clear();  Console.WriteLine("Enter train number:");  trainNumber = int.Parse(Console.ReadLine());  }   while (departure == DateTime.MinValue)  {  //try-catch blocks to make sure that user have   //entered valid value. When value is invalid,   //user gets asked again  try  {  Console.Clear();  Console.WriteLine("Enter train departure time:");  departure = DateTime.Parse(Console.ReadLine());  }  catch  {  Console.WriteLine("The input string was not in the correct format. Try again");  continue;  }  }   return new TrainAsClass(destination, trainNumber, departure);  }   private static bool AreSelectedTrainsEqual(IEnumerable<ITrain> trains)  {  ITrain[] trainsArray = trains.ToArray();   if (trainsArray.Count() < 2)  throw new Exception("Not enough trains entered");   WriteTrainsToConsole(trainsArray);   Console.WriteLine("Select the comparable train");  var comparable = int.Parse(Console.ReadLine());   Console.WriteLine("Select the train to compare");  var toCompare = int.Parse(Console.ReadLine());   return trainsArray[comparable - 1].Equals(trainsArray[toCompare - 1]);  }   //method to list all available Train class and structure instances   private static void WriteTrainsToConsole(IEnumerable<ITrain> trains)  {  Console.Clear();  var trainsArray = trains.ToArray();   for (int i = 1; i <= trains.Count(); i++)  }  Console.WriteLine($"{i}\t{trainsArray[i - 1].ToString()}");  }  Console.ReadLine();  }  } |

1. **Описание программы:**

Учитывая, что в языке C# структуры отличаются от классов основным образом тем, что они передаются по значению, а классы по ссылке. Следовательно, отличий, описанных в методических указаниях для языка С++, в языке С# нет. Для выполнения лабораторной работы было принято решение организовать программу таким образом, чтобы показать отличия и схожести структур и классов в языке C#.

Тем не менее, для соответствия лабораторной работы условию поля структуры были сделаны get-set свойствами для создания подобия такого подхода подходу языка С++. Поля класса были сделаны get-only свойствами для того, чтобы объекты класса приходилось инициализировать через конструктор по той же причине.

В программе с помощью интерфейса была реализована возможность хранить объекты типа Train (как структуры, так и классы) в одной коллекции. Кроме того, объекты типа Train с реализацией-классом и реализацией-структурой можно сравнивать между собой.

1. **Тестовые примеры:**

Программа должна вывести, что результат сравнения объектов классов и структур с одинаковым значением времени отправления - равенство двух объектов. Проверим это. Создадим класс и инициализируем его таким образом:

* номер поезда = 1;
* пункт назначения - Moscow;
* время отправления = 12:00.  
  Зададим структуру с таким же временем отправления:
* номер поезда = 2;
* пункт назначения = sevastopol;
* время отправления = 12:00.

Проверим работу программы:

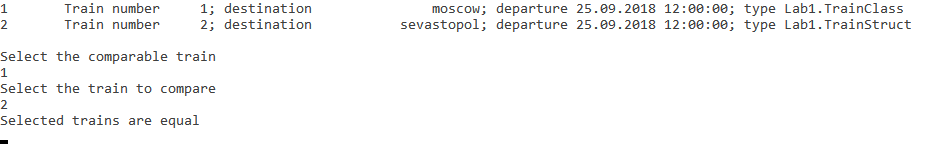


Рис. 1 - Результат работы программы над первым тестовым примером

Для поездов с разными значениями времени отправления программа выводит неравенство объектов-поездов:

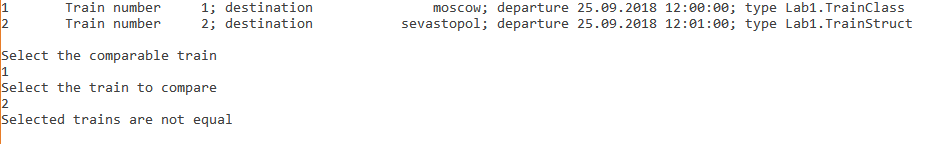


Рис 2 - Результат работы программы над вторым тестовым примером

1. **Вывод:** в ходе лабораторной работы были изучены отличия между объектным и структурным подходом объектно-ориентированного программирования. Кроме того, были изучены схожести и отличия между классами и структурами в языках С++ и С#.