Дисциплина «Программирование корпоративных систем» Рабочая тетрадь 6

Делегаты, лямбда-выражения, события

|  |  |
| --- | --- |
| Ознакомьтесь с теоретическим материалом лекции №6 | |
| **Задание 1** | |
| ***Задача:*** | |
|  | **Использование делегатов для сортировки массива**  Напишите программу, которая сортирует массив строк по разным критериям с использованием делегатов. Программа должна предложить пользователю выбрать один из следующих способов сортировки:   * По длине строки (от самой короткой до самой длинной). * По алфавиту. * По количеству гласных букв в строке.   Создайте соответствующий делегат и передайте его в метод сортировки.  Сопроводите решение текстовым объяснением использования вами механизма делегатов. |
| ***Решение:*** | |
|  | using System; using System.Collections.Generic; using System.Linq;  class Program {  // Определение делегата для сравнения двух объектов типа T  public delegate int Comparison<T>(T x, T y);   static void *Main*(string[] args)  {  string[] strings = { "Sunshine", "apple", "happiness", "adventure" };   Console.*WriteLine*("Выберите критерий сортировки:");  Console.*WriteLine*("1. По длине строки");  Console.*WriteLine*("2. По алфавиту");  Console.*WriteLine*("3. По количеству гласных букв");   int choice = Convert.*ToInt32*(Console.*ReadLine*());   switch (choice)  {  case 1:  *Sort*(strings, *CompareByLen*);   break;  case 2:  *Sort*(strings, *CompareByAlf*);   break;  case 3:  *Sort*(strings, *CompareByContGlas*);   break;  default:  Console.*WriteLine*("Неверный выбор.");   return;  }   Console.*WriteLine*("Отсортированный массив:");  foreach (string s in strings)  {  Console.*WriteLine*(s);  }  }   // Метод сортировки, принимающий массив и делегат для сравнения  static void *Sort*<T>(T[] array, Comparison<T> comparison)  {  // Сортировка массива с использованием ComparisonComparer  Array.*Sort*(array, new ComparisonComparer<T>(comparison));  }   // Класс, реализующий интерфейс IComparer<T> для использования делегата  class ComparisonComparer<T> : IComparer<T>  {  private readonly Comparison<T> \_comparer; // Хранит делегат для сравнения   // Конструктор принимает делегат сравнения  public ComparisonComparer(Comparison<T> comparer)  {  \_comparer = comparer;  }   // Метод сравнения  public int Compare(T x, T y)  {  return \_comparer(x, y); // Вызов делегата  }  }   // Функция сравнения по длине строки  static int *CompareByLen*(string x, string y)  {  return x.Length.CompareTo(y.Length);   }   // Функция сравнения по алфавиту  static int *CompareByAlf*(string x, string y)  {  return String.*Compare*(x, y);   }   // Функция сравнения по количеству гласных букв  static int *CompareByContGlas*(string x, string y)  {  char[] vowels = { 'a', 'e', 'i', 'o', 'u' };   int countX = *CountVowels*(x, vowels);   int countY = *CountVowels*(y, vowels);   return countX.CompareTo(countY);   }   // Метод для подсчета гласных в строке  static int *CountVowels*(string str, char[] vowels)  {  return str.Count(c => vowels.Contains(char.*ToLowerInvariant*(c)));  } } |
| ***Ответ:*** | |
|  |  |
| **Задание 2** | |
| ***Задача:*** | |
|  | **Фильтрация строк с использованием лямбда-выражений**  Напишите программу, которая фильтрует строки в массиве по указанным пользователем критериям с использованием лямбда-выражений. Программа должна предложить пользователю выбрать один из следующих вариантов фильтрации:   * Строки, длина которых больше указанного числа. * Строки, начинающиеся с определенной буквы. * Строки, содержащие заданное количество гласных букв или больше.   Реализуйте фильтрацию с помощью лямбда-выражений.  Сопроводите решение текстовым объяснением использования вами механизма лямбда-выражений. |
| ***Решение:*** | |
|  | using System; using System.Collections.Generic; using System.Linq;  class Program {  static void *Main*()  {  string[] strings = { "Sunshine", "apple", "happiness", "like" };   Console.*WriteLine*("Выберите критерий фильтрации:");  Console.*WriteLine*("1. Строки, длина которых больше указанного числа.");  Console.*WriteLine*("2. Строки, начинающиеся с определенной буквы.");  Console.*WriteLine*("3. Строки, содержащие заданное количество гласных букв или больше.");   int choice = Convert.*ToInt32*(Console.*ReadLine*());   List<string> filteredStrings = null;   switch (choice)  {  case 1:  Console.*Write*("Введите длину: ");  int length = Convert.*ToInt32*(Console.*ReadLine*());  filteredStrings = *GetStringsByLength*(strings, length);  break;  case 2:  Console.*Write*("Введите букву: ");  char letter = Convert.*ToChar*(Console.*ReadLine*().ToUpper());  filteredStrings = *GetStringsByFirstLetter*(strings, letter);  break;  case 3:  Console.*Write*("Введите минимальное количество гласных: ");  int vowelCount = Convert.*ToInt32*(Console.*ReadLine*());  filteredStrings = *GetStringsByVowelCount*(strings, vowelCount);  break;  default:  Console.*WriteLine*("Неверный выбор.");  return;  }   if (filteredStrings != null && filteredStrings.Count > 0)  {  Console.*WriteLine*("Отфильтрованные строки:");  foreach (var str in filteredStrings)  {  Console.*WriteLine*(str);   }  }  else  {  Console.*WriteLine*("Нет подходящих строк.");  }  }   // Метод для фильтрации строк по длине  static List<string> *GetStringsByLength*(string[] strings, int minLength)  {  return strings.Where(str => str.Length > minLength).ToList();  }   // Метод для фильтрации строк по первой букве  static List<string> *GetStringsByFirstLetter*(string[] strings, char firstLetter)  {  return strings.Where(str => Char.*ToUpper*(str[0]) == firstLetter).ToList();  }   // Метод для фильтрации строк по количеству гласных  static List<string> *GetStringsByVowelCount*(string[] strings, int minVowelCount)  {  char[] vowels = { 'A', 'E', 'I', 'O', 'U' };  return strings.Where(str => *GetVowelCount*(str, vowels) >= minVowelCount).ToList();  }   // Метод для подсчета гласных в строке  static int *GetVowelCount*(string str, char[] vowels)  {  return str.Count(c => vowels.Contains(Char.*ToUpper*(c)));  } } |
| ***Ответ:*** | |
|  |  |

|  |  |
| --- | --- |
| **Задание 3** | |
| ***Задача:*** | |
|  | **Напишите программу, которая моделирует работу термостата.**  Устройство отслеживает изменение температуры и генерирует события, если:   * Температура превышает заданное максимальное значение. * Температура падает ниже заданного минимального значения.   Программа должна:   * Позволить пользователю задать минимальную и максимальную температуры. * Реализовать два обработчика событий: один для предупреждения о перегреве, другой — о переохлаждении. * Симулировать изменение температуры в случайных пределах.   Сопроводите решение текстовым объяснением использования вами механизма событий. |
| ***Решение:*** | |
|  | using System;  public class Thermostat {  // Событие при превышении максимального предела  public event EventHandler MaxTemperatureExceeded;    // Событие при падении ниже минимального предела  public event EventHandler MinTemperatureFallenBelow;   private double \_minTemperature;   private double \_maxTemperature;   private double \_currentTemperature;    // Конструктор для установки минимальной и максимальной температуры  public Thermostat(double minTemp, double maxTemp)  {  \_minTemperature = minTemp;  \_maxTemperature = maxTemp;  }   // Свойство для работы с текущей температурой  public double CurrentTemperature  {  get { return \_currentTemperature; }  set  {  \_currentTemperature = value;   ValidateTemperature(); // Проверка текущей температуры  }  }   // Метод для проверки текущей температуры  private void ValidateTemperature()  {  if (\_currentTemperature > \_maxTemperature)  {  RaiseMaxTemperatureExceeded(EventArgs.*Empty*); // Вызов события превышения максимальной температуры  }  else if (\_currentTemperature < \_minTemperature)  {  RaiseMinTemperatureFallenBelow(EventArgs.*Empty*); // Вызов события падения ниже минимальной температуры  }  }   // Метод для обработки превышения максимальной температуры  protected virtual void RaiseMaxTemperatureExceeded(EventArgs e)  {  MaxTemperatureExceeded?.Invoke(this, e);   }   // Метод для обработки падения температуры ниже минимума  protected virtual void RaiseMinTemperatureFallenBelow(EventArgs e)  {  MinTemperatureFallenBelow?.Invoke(this, e);   } }  public class Program {  public static void *Main*()  {  Console.*Write*("Введите минимальную температуру: ");  double minTemp = Convert.*ToDouble*(Console.*ReadLine*());    Console.*Write*("Введите максимальную температуру: ");  double maxTemp = Convert.*ToDouble*(Console.*ReadLine*());   var thermostat = new Thermostat(minTemp, maxTemp);   // Подписка на события превышения и падения температуры  thermostat.MaxTemperatureExceeded += *HandleMaxTemperatureWarning*;  thermostat.MinTemperatureFallenBelow += *HandleMinTemperatureWarning*;   Random random = new Random();   while (true)   {  double tempChange = (random.NextDouble() \* 10) - 5;   thermostat.CurrentTemperature += tempChange;    Console.*WriteLine*($"Текущая температура: {thermostat.CurrentTemperature}");   System.Threading.Thread.*Sleep*(1000);  }  }   // Метод для предупреждения о превышении максимальной температуры  private static void *HandleMaxTemperatureWarning*(object sender, EventArgs e)  {  Console.*WriteLine*("Внимание! Температура превысила допустимый максимум!");  }   // Метод для предупреждения о падении температуры ниже минимального уровня  private static void *HandleMinTemperatureWarning*(object sender, EventArgs e)  {  Console.*WriteLine*("Внимание! Температура опустилась ниже допустимого минимума!");  } } |
| ***Ответ:*** | |

