# Лабораторная работа №10

## Массивы и коллекции

### Вариант 1

Цель работы: приобретение практических навыков работы с типами массивов и коллекций в .NET Framework.

### Задание к работе

1. Создать класс, содержащий заданные в соответствии с вариантом имя и поля. Определить необходимые свойства для доступа к полям. Реализовать интерфейс IComparable и его обобщенный вариант.
2. Определить класс, основанный на структуре данных типа «односвязный линейный список», который позволяет хранить объекты заданного класса. Реализовать интерфейсы IList, ICollection, IEnumerable и их обобщенные варианты.
3. Добавить методы для добавления данных в список и удаления элементов из списка.
4. Добавить метод для поиска в соответствии с условием, представленным в задаче. Если элемент с заданным условием не найден, метод должен сгенерировать исключение.
5. Написать программу для решения поставленной задачи, используя созданные классы.

### Задание для первого варианта

1. Определить класс с именем Student, содержащий следующие поля:
   * фамилия и инициалы;
   * номер группы;
   * успеваемость (массив из пяти элементов).
2. Написать программу, выполняющую следующие действия:
   * ввод с клавиатуры данных в массив, состоящий из объектов типа Student; записи должны быть упорядочены по возрастанию номера группы;
   * вывод на экран фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0;
   * если таких студентов нет, вывести соответствующее сообщение.

# Решение

В классе Student реализованы следующие поля:

* Name — строковое поле для хранения имени и фамилии студента;
* Group — целочисленное поле для хранения номера группы студента;
* Marks — целочисленный массив из пяти элементов для хранения оценок студента.

## List.cs

|  |  |
| --- | --- |
| 001 | **using** System; |
| 002 | **using** System.Collections.Generic; |
| 003 | **using** System.Linq; |
| 004 | **using** System.Text; |
| 005 | **using** System.Threading.Tasks; |
| 006 | **using** System.Collections; |
| 007 |  |
| 008 | **namespace** lab |
| 009 | { |
| 010 | **class** ListElement |
| 011 | { |
| 012 | **public** Student Data { **get**; **set**; } |
| 013 | **public** ListElement Next { **get**; **set**; } |
| 014 |  |
| 015 | **public** ListElement() |
| 016 | { |
| 017 | Data = **null**; |
| 018 | Next = **null**; |
| 019 | } |
| 020 | } |
| 021 |  |
| 022 | **class** List : IList, IList<Student> |
| 023 | { |
| 024 | ListElement Start; |
| 025 | ListElement Current; |
| 026 | **public** **int** Count { **get**; **private** **set**; } |
| 027 |  |
| 028 | **public** List() |
| 029 | { |
| 030 | Start = Current = **new** ListElement(); |
| 031 | **this**.Count = 0; |
| 032 | } |
| 033 |  |
| 034 | **public** **void** Search(**double** avg) |
| 035 | { |
| 036 | **try** |
| 037 | { |
| 038 | **int** counter = 0; |
| 039 | **for** (ListElement a = Start.Next; a != **null**; a = a.Next) |
| 040 | { |
| 041 | **if** (a.Data.Avg > avg) |
| 042 | { |
| 043 | Console.WriteLine(a.Data.ToString()); |
| 044 | counter++; |
| 045 | } |
| 046 | } |
| 047 | **if** (counter != 0) |
| 048 | { |
| 049 | **return**; |
| 050 | } |
| 051 | **throw** **new** Exception(*"Искомые студенты не найдены."*); |
| 052 | } |
| 053 | **catch** (Exception e) |
| 054 | { |
| 055 | Console.WriteLine(e.Message); |
| 056 | } |
| 057 | } |
| 058 |  |
| 059 | **public** **int** SearchIndex(Student item) |
| 060 | { |
| 061 | **int** i = 0; |
| 062 | **for** (; i < Count; i++) |
| 063 | { |
| 064 | **if** (**this**[i].CompareTo(item) == -1) |
| 065 | { |
| 066 | **return** i; |
| 067 | } |
| 068 | } |
| 069 | **return** i; |
| 070 | } |
| 071 |  |
| 072 | **public** **int** IndexOf(Student item) |
| 073 | { |
| 074 | **int** i = 0; |
| 075 | ListElement pos; |
| 076 |  |
| 077 | **for** (pos = Start.Next; (pos != **null**) && pos.Data.Equals(item);   pos = pos.Next) |
| 078 | { |
| 079 | i++; |
| 080 | } |
| 081 |  |
| 082 | **if** (pos == **null**) |
| 083 | { |
| 084 | **return** -1; |
| 085 | } |
| 086 | **else** |
| 087 | { |
| 088 | **return** i; |
| 089 | } |
| 090 | } |
| 091 |  |
| 092 | **int** IList.IndexOf(**object** value) |
| 093 | { |
| 094 | **return** IndexOf((Student)value); |
| 095 | } |
| 096 |  |
| 097 | **public** **void** Insert(**int** index, Student item) |
| 098 | { |
| 099 | **if** (**this**.IsReadOnly) **throw** **new** NotSupportedException(); |
| 100 | **if** ((index > -1) && (index <= Count)) |
| 101 | { |
| 102 | ListElement t = Current; |
| 103 | Current = Start; |
| 104 | **for** (**int** i = 0; i <= (index - 1); i++) |
| 105 | { |
| 106 | Current = Current.Next; |
| 107 | } |
| 108 | **this**.Add(item); |
| 109 | Current = t; |
| 110 | **return**; |
| 111 | } |
| 112 | **throw** **new** ArgumentOutOfRangeException(); |
| 113 | } |
| 114 |  |
| 115 | **void** IList.Insert(**int** index, **object** value) |
| 116 | { |
| 117 | **this**.Insert(index, (Student)value); |
| 118 | } |
| 119 |  |
| 120 | **public** **void** RemoveAt(**int** index) |
| 121 | { |
| 122 | **if** (**this**.IsReadOnly) **throw** **new** NotSupportedException(); |
| 123 | **if** (index > -1 && index < Count) |
| 124 | { |
| 125 | ListElement t = **this**.Start; |
| 126 | **for** (**int** i = 0; i < index - 1; i++) |
| 127 | { |
| 128 | t = t.Next; |
| 129 | } |
| 130 | t.Next = t.Next.Next; |
| 131 | Count--; |
| 132 | } |
| 133 | **throw** **new** ArgumentOutOfRangeException(); |
| 134 | } |
| 135 |  |
| 136 | **void** IList.RemoveAt(**int** index) |
| 137 | { |
| 138 | **this**.RemoveAt(index); |
| 139 | } |
| 140 |  |
| 141 | **public** Student **this**[**int** index] |
| 142 | { |
| 143 | **get** |
| 144 | { |
| 145 | **if** ((index > -1) && (index < Count)) |
| 146 | { |
| 147 | ListElement t = Start.Next; |
| 148 | **for** (**int** i = 0; i < index; i++) |
| 149 | { |
| 150 | t = t.Next; |
| 151 | } |
| 152 | **return** t.Data; |
| 153 | } |
| 154 | **throw** **new** ArgumentOutOfRangeException(); |
| 155 | } |
| 156 | **set** |
| 157 | { |
| 158 | **if** ((index > 0) && (index < Count)) |
| 159 | { |
| 160 | Insert(index, value); |
| 161 | } |
| 162 | **throw** **new** ArgumentOutOfRangeException(); |
| 163 | } |
| 164 | } |
| 165 |  |
| 166 | **object** IList.**this**[**int** index] |
| 167 | { |
| 168 | **get** |
| 169 | { |
| 170 | **return** **this**[index]; |
| 171 | } |
| 172 | **set** |
| 173 | { |
| 174 | **this**[index] = (Student)value; |
| 175 | } |
| 176 | } |
| 177 |  |
| 178 | **public** **void** Add(Student item) |
| 179 | { |
| 180 | ListElement newListElement = **new** ListElement(); |
| 181 | newListElement.Next = **this**.Current.Next; |
| 182 | **this**.Current.Next = newListElement; |
| 183 | newListElement.Data = item; |
| 184 | Count++; |
| 185 |  |
| 186 | **int** IList.Add(**object** value) |
| 187 | { |
| 188 | **this**.Add((Student)value); |
| 189 | **return** IndexOf((Student)value); |
| 190 | } |
| 191 |  |
| 192 | **public** **void** Clear() |
| 193 | { |
| 194 | **for** (ListElement t = **this**.Start; t != **null**; t = t.Next) |
| 195 | { |
| 196 | t = **null**; |
| 197 | } |
| 198 | } |
| 199 |  |
| 200 | **void** IList.Clear() |
| 201 | { |
| 202 | **this**.Clear(); |
| 203 | } |
| 204 |  |
| 205 | **public** **bool** Contains(Student item) |
| 206 | { |
| 207 | **return** IndexOf(item) != -1; |
| 208 | } |
| 209 |  |
| 210 | **bool** IList.Contains(**object** value) |
| 211 | { |
| 212 | **return** **this**.Contains((Student)value); |
| 213 | } |
| 214 |  |
| 215 | **public** **void** CopyTo(Student[] array, **int** arrayIndex) |
| 216 | { |
| 217 | **if** (array == **null**) |
| 218 | { |
| 219 | **throw** **new** ArgumentNullException(); |
| 220 | } |
| 221 |  |
| 222 | **if** (arrayIndex < 0) |
| 223 | { |
| 224 | **throw** **new** ArgumentOutOfRangeException(); |
| 225 | } |
| 226 |  |
| 227 | **if** (**this**.Count + arrayIndex > array.Length) |
| 228 | { |
| 229 | **throw** **new** ArgumentException(); |
| 230 | } |
| 231 |  |
| 232 | **int** i = 0; |
| 233 | **while** (i < **this**.Count) |
| 234 | { |
| 235 | array[arrayIndex++] = **this**[i++]; |
| 236 | } |
| 237 | } |
| 238 |  |
| 239 | **void** ICollection.CopyTo(Array array, **int** index) |
| 240 | { |
| 241 | **this**.CopyTo((Student[])array, index); |
| 242 | } |
| 243 |  |
| 244 | **int** ICollection.Count |
| 245 | { |
| 246 | **get** { **return** **this**.Count; } |
| 247 | } |
| 248 |  |
| 249 | **public** **bool** IsReadOnly |
| 250 | { |
| 251 | **get** { **return** **false**; } |
| 252 | } |
| 253 |  |
| 254 | **bool** IList.IsReadOnly |
| 255 | { |
| 256 | **get** { **return** **this**.IsReadOnly; } |
| 257 | } |
| 258 |  |
| 259 | **public** **bool** Remove(Student item) |
| 260 | { |
| 261 | ListElement pos; |
| 262 | **for** (pos = **this**.Start; pos.Next != **null**  && pos.Next.Data.Equals(item); pos = pos.Next); |
| 263 | **if** (pos != **null**) |
| 264 | { |
| 265 | pos.Next = pos.Next.Next; |
| 266 | **return** **true**; |
| 267 | } |
| 268 | **else** |
| 269 | { |
| 270 | **return** **false**; |
| 271 | } |
| 272 | } |
| 273 |  |
| 274 | **void** IList.Remove(**object** value) |
| 275 | { |
| 276 | **this**.Remove((Student)value); |
| 277 | } |
| 278 |  |
| 279 | **public** IEnumerator<Student> GetEnumerator() |
| 280 | { |
| 281 | ListElement startPos = **this**.Start; |
| 282 |  |
| 283 | **while** (startPos.Next != **null**) |
| 284 | { |
| 285 | **yield** **return** startPos.Next.Data; |
| 286 | startPos = startPos.Next; |
| 287 | } |
| 288 | } |
| 289 |  |
| 290 | IEnumerator IEnumerable.GetEnumerator() |
| 291 | { |
| 292 | **return** GetEnumerator(); |
| 293 | } |
| 294 |  |
| 295 | **bool** IList.IsFixedSize |
| 296 | { |
| 297 | **get** { **return** **false**; } |
| 298 | } |
| 299 |  |
| 300 | **bool** ICollection.IsSynchronized |
| 301 | { |
| 302 | **get** { **return** **false**; } |
| 303 | } |
| 304 |  |
| 305 | **object** ICollection.SyncRoot |
| 306 | { |
| 307 | **get** { **return** **this**[0]; } |
| 308 | } |
| 309 | } |
| 310 | } |

## Student.cs

|  |  |
| --- | --- |
| 01 | **using** System; |
| 02 | **using** System.Collections.Generic; |
| 03 | **using** System.Linq; |
| 04 | **using** System.Text; |
| 05 | **using** System.Threading.Tasks; |
| 06 |  |
| 07 | **namespace** lab |
| 08 | { |
| 09 | **class** Student : IComparable, IComparable<Student> |
| 10 | { |
| 11 | **public** **string** Name { **get**; **set**; } |
| 12 | **public** **int** Group { **get**; **set**; } |
| 13 | **private** **int**[] marks; |
| 14 | **public** **int**[] Marks |
| 15 | { |
| 16 | **get** |
| 17 | { |
| 18 | **return** marks; |
| 19 | } |
| 20 | **set** |
| 21 | { |
| 22 | marks = value; |
| 23 | Avg = 0; |
| 24 | **foreach** (**int** mark **in** marks) |
| 25 | { |
| 26 | Avg += mark; |
| 27 | } |
| 28 | Avg /= 5; |
| 29 | } |
| 30 | } |
| 31 | **public** **float** Avg { **get**; **set**; } |
| 32 |  |
| 33 | **public** Student() |
| 34 | { |
| 35 | Marks = **new** **int**[5]; |
| 36 | } |
| 37 |  |
| 38 | **public** Student(**string** name, **int** group, **int**[] marks) |
| 39 | { |
| 40 | Name = name; |
| 41 | Group = group; |
| 42 | Marks = marks; |
| 43 | Avg = 0; |
| 44 | **foreach** (**int** mark **in** marks) |
| 45 | { |
| 46 | Avg += mark; |
| 47 | } |
| 48 | Avg /= 5; |
| 49 | } |
| 50 |  |
| 51 | **public** **override bool** Equals(**object** obj) |
| 52 | { |
| 53 | **var** a = (Student)obj; |
| 54 | **return** Name.Equals(a.Name) && Group.Equals(a.Group)  && Marks.Equals(a.Marks); |
| 55 | } |
| 56 |  |
| 57 | **public** **int** CompareTo(Student obj) |
| 58 | { |
| 59 | **if** (Group < obj.Group) |
| 60 | { |
| 61 | **return** 1; |
| 62 | } |
| 63 | **else** **if** (Group > obj.Group) |
| 64 | { |
| 65 | **return** -1; |
| 66 | } |
| 67 | **else** |
| 68 | { |
| 69 | **return** 0; |
| 70 | } |
| 71 | } |
| 72 |  |
| 73 | **public** **override string**  ToString() |
| 74 | { |
| 75 | **return** String.Format(*"Имя: {0}\tГруппа: {1}\tОценки: {2} {3}   {4} {5} {6}",* Name, Group, Marks[0], Marks[1],   Marks[2], Marks[3], Marks[4]); |
| 76 | } |
| 77 |  |
| 78 | **int** IComparable.CompareTo(**object** obj) |
| 79 | { |
| 80 | **return** CompareTo((Student)obj); |
| 81 | } |
| 82 | } |
| 83 | } |

## Program.cs

|  |  |
| --- | --- |
| 01 | **using** System; |
| 02 | **using** System.Collections.Generic; |
| 03 | **using** System.Linq; |
| 04 | **using** System.Text; |
| 05 | **using** System.Threading.Tasks; |
| 06 | **using** System.Threading; |
| 07 |  |
| 08 | **namespace** lab |
| 09 | { |
| 10 | **class** Program |
| 11 | { |
| 12 | **static** **void** Main(**string**[] args) |
| 13 | { |
| 14 | Thread.CurrentThread.CurrentCulture =  **new** System.Globalization.CultureInfo(*"ru-RU"*); |
| 15 | Thread.CurrentThread.CurrentCulture  .NumberFormat.NumberDecimalSeparator = *"."*; |
| 16 |  |
| 17 | Console.Write(*"Введите количество записей: "*); |
| 18 | **int** n = Convert.ToInt32(Console.ReadLine()); |
| 19 | List List = **new** List(); |
| 20 | **for** (**int** i = 0; i < n; i++) |
| 21 | { |
| 22 | Student tmp = **new** Student(); |
| 23 | Console.Write(*"{0}) Имя: "*, i + 1); |
| 24 | tmp.Name = Console.ReadLine(); |
| 25 |  |
| 26 | Console.Write(*" Группа: "*); |
| 27 | tmp.Group = Convert.ToInt32(Console.ReadLine()); |
| 28 |  |
| 29 | Console.Write(*" Пять оценок через пробел: "*); |
| 30 | **string** [] marksString = Console.ReadLine().Split(*' '*); |
| 31 | **int**[] marks = **new** **int**[5]; |
| 32 | **for** (**int** j = 0; j < 5; j++) |
| 33 | { |
| 34 | marks[j] = Int32.Parse(marksString[j]); |
| 35 | } |
| 36 | tmp.Marks = marks; |
| 37 |  |
| 38 | List.Insert(List.SearchIndex(tmp), tmp); |
| 39 | } |
| 40 |  |
| 41 | Console.Clear(); |
| 42 | **foreach** (Student tmp **in** List) |
| 43 | { |
| 44 | Console.WriteLine(tmp.ToString()); |
| 45 | } |
| 46 |  |
| 47 | Console.Write(*"\nВывести список студентов с оценкой более: "*); |
| 48 | **double** mark = Convert.ToDouble(Console.ReadLine()); |
| 49 | Console.WriteLine(); |
| 50 |  |
| 51 | List.Search(mark); |
| 52 | Console.ReadKey(); |
| 53 | } |
| 54 | } |
| 55 | } |