Отчёт ЛР№4 Понкращенков Д.Б. ПИН-21Д.

Вариант №1.

Оглавление

1. Задание

Определить новые версии классов **Student** и **StudentCollection<TKey>** из лабораторной работы 3.

Новая версия класса **Student** реализует интерфейс **System.ComponentModel.INotifyPropertyChanged**. Событие **PropertyChanged** из интерфейса System.ComponentModel.INotifyPropertyChanged происходит при изменении значений свойств класса **Student**, связанных с номером группы и формой обучения. Название свойства, значение которого изменилось, событие PropertyChanged передает своим обработчикам через свойство PropertyName класса PropertyChangedEventArgs.
Для информации о типе изменений, которые произошли в коллекциях, определить перечисление (enum) **Action** со значениями **Add**, **Remove** и **Property**.

Для события, которое бросают методы класса StudentCollection<TKey>, определить делегат **StudentsChangedHandler<TKey>** с сигнатурой:

```
void StudentsChangedHandler<TKey>
  (object source, StudentsChangedEventArgs<TKey> args);
```

Класс **StudentsChangedEventArgs<TKey>**, производный от класса System.EventArgs, содержит

- открытое автореализуемое свойство типа string с названием коллекции;
- открытое автореализуемое свойство типа Action с информацией о том, чем вызвано событие, - удалением элемента, добавлением элемента или изменением данных элемента;
- открытое автореализуемое свойство типа string с названием свойства класса Student, которое является источником изменения данных элемента; для событий, брошенных при удалении или добавлении элемента, значение свойства - пустая строка;
- открытое автореализуемое свойство типа TKey с ключом добавленного, удаленного или измененного элемента;
- конструктор с параметрами типа string, Action, string и TKey для инициализации значений всех свойств класса;
- перегруженную версию метода string ToString().

В новую версию класса **StudentCollection<TKey>** добавить

- открытое автореализуемое свойство типа string с названием коллекции;
- метод bool Remove(Student st) для удаления элемента со значением st из словаря Dictionary<TKey, Student>; если в словаре нет элемента st, метод возвращает значение false;
- событие **StudentsChanged** типа **StudentsChangedHandler<TKey>**, которое происходит, когда в коллекцию добавляются элементы, из нее удаляется элемент или изменяются данные одного из ее элементов.

Определить класс **Journal**, который можно использовать для накопления информации об изменениях в коллекциях типа StudentCollection<TKey>. Класс Journal хранит информацию об изменениях в коллекциях в списке объектов типа **JournalEntry**. Класс JournalEntry содержит информацию об отдельном изменении, которое произошло в коллекциях.

Класс **JournalEntry** содержит автореализуемые свойства

- типа string с названием коллекции;
- типа Action с информацией о типе события;
- типа string с названием свойства класса Student, которое явилось причиной изменения данных элемента;
- типа string с текстовым представлением ключа добавленного, удаленного или измененного элемента;
- конструктор для инициализации всех свойств класса;
- перегруженную версию метода string ToString().

Класс **Journal** содержит

- закрытое поле типа System.Collections.Generic.List<JournalEntry>;
- обработчик события StudentsChanged, который на основе информации из объекта StudentsChangedEventArgs, создает элемент JournalEntry и добавляет его в список List<JournalEntry>;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<JournalEntry>.

В методе **Main()**

1. Создать две коллекции StudentCollection<string> с разными названиями.
2. Создать объект Journal и подписать его на события StudentsChanged из обеих коллекций StudentCollection<string>.
3. Внести изменения в коллекции StudentCollection<string>:
   - добавить элементы Student в коллекции;
   - изменить значения разных свойств элементов, входящих в коллекцию;
   - удалить элемент из коллекции;
   - изменить данные в удаленном элементе.
4. Вывести данные объекта Journal.

2. Результат работы программы

Выполнение задания по вариантам:

```
Рандомизация коллекций 1 и 2...

Изменение параметров элементов коллекций 1 и 2...

Удаление элементов коллекций 1 и 2...

Изменение параметров удаленного элемента коллекций 1 и 2...

Печать Journal:
- studentCollection1 Add  1
- studentCollection1 Add  2
- studentCollection1 Add  3
- studentCollection2 Add  4
- studentCollection2 Add  5
- studentCollection2 Add  6
- studentCollection2 Add  7
- studentCollection1 Property education 3
- studentCollection2 Property education 6
- studentCollection1 Property numberGroup 3
- studentCollection2 Property numberGroup 6
- studentCollection1 Remove  3
- studentCollection2 Remove  7
- studentCollection2 Property education 6
- studentCollection2 Property numberGroup 6
```

3. Листинг

```
1. //-----------------------------------------------------------------------------||
2. // Файл Main.cs
3. //-----------------------------------------------------------------------------||
4.
5. using LR2;
6. using System.Security.Cryptography;
7.
8. namespace LR4
9. {
10.     internal class Program
11.     {
12.
13.         private static Random random = new Random();
14.
15.         static void Main(string[] args)
16.         {
17.             int countElem = 0;
18.             var studentCollection1 = new StudentCollection<string>("studentCollection1",
    (Student st) => { return st.Forename + ++countElem; });
19.             var studentCollection2 = new StudentCollection<string>("studentCollection2",
    (Student st) => { return st.Forename + ++countElem; });
20.
21.             Journal<string> journal = new Journal<string>();
22.
23.             studentCollection1.StudentsChanged += journal.StudentsChangedHandler;
24.             studentCollection2.StudentsChanged += journal.StudentsChangedHandler;
25.
26.             Console.WriteLine("\n Рандомизация коллекций 1 и 2...");
27.
28.             var toDel1 = getStudentRND();
29.             var toDel2 = getStudentRND();
30.
31.             studentCollection1.AddDefaults();
32.             studentCollection1.AddStudents([
33.                 getStudentRND(),
34.                 toDel1
35.             ]);
36.
37.             studentCollection2.AddDefaults();
38.             studentCollection2.AddStudents([
39.                 getStudentRND(),
40.                 toDel1,
41.                 toDel2
42.             ]);
43.
44.             Console.WriteLine("\n Изменение параметров элементов коллекций 1 и 2...");
45.             toDel1.Education = Education.Bachelor;
46.             toDel1.NumberGroup = 99;
47.
48.             Console.WriteLine("\n Удаление элементов коллекций 1 и 2...");
49.             studentCollection1.Remove(toDel1);
50.             studentCollection2.Remove(toDel2);
51.
52.             Console.WriteLine("\n Изменение параметров удаленного элемента коллекций 1 и
    2...");
53.             toDel1.Education = Education.Specialist;
54.             toDel1.NumberGroup = 120;
55.
56.             Console.WriteLine("\n Печать Journal:");
57.             Console.WriteLine(journal.ToString());
58.         }
```

```
        #region --- Рандомайзеры ---

        static Student getStudentRND()
        {
            var student = new Student();

            student.NumberGroup = RandomNumberGenerator.GetInt32(100, 200);
            student.Education = (Education)RandomNumberGenerator.GetInt32(0, 3);

            return student;
        }

        static string getStringRND()
        {
            int length = 6;
            const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
            return new string(Enumerable.Repeat(chars, length)
                .Select(s => s[random.Next(s.Length)]).ToArray());
        }

        #endregion
    }
}
```

```
//-----------------------------------------------------------------------------||
// Файл EAction.cs
//-----------------------------------------------------------------------------||
namespace LR4
{
    public enum EAction
    {
         Add,
         Remove,
         Property
    }
}
```

```
//-----------------------------------------------------------------------------||
// Файл StudentsChangedEventArgs.cs
//-----------------------------------------------------------------------------||
namespace LR4
{
     internal class StudentsChangedEventArgs<TKey> : System.EventArgs
     {
          public string NameCollection { get; set; }
          public string PropertyStudent {  get; set; }
          public EAction Action { get; set; }
          public TKey Key { get; set; }

          public StudentsChangedEventArgs(string nameCollection, EAction action, string
          property, TKey key)
          {
               NameCollection = nameCollection;
               PropertyStudent = property;
               Action = action;
               Key = key;
          }
          public override string ToString()
          {
               return NameCollection + " " + Action.ToString() + " " + PropertyStudent + "
               " + Key;
          }
     }
}
```

```
1.    //-----------------------------------------------------------------------------------------||
2.    // Файл Journal.cs
3.    //-----------------------------------------------------------------------------------------||

4.    using System.Text;

5.    namespace LR4
6.    {
7.         internal class Journal<TKey>
8.         {
9.              List<JournalEntry> journalEntries = new List<JournalEntry>();

10.             /// <summary>
11.             /// Обработчик событий
12.             /// </summary>
13.             /// <param name="source"></param>
14.             /// <param name="args"></param>
15.             public void StudentsChangedHandler(object source, StudentsChangedEventArgs<TKey>
                args)
16.             {
17.                  var tmp = new JournalEntry(args.NameCollection, args.Action,
                     args.PropertyStudent, args.Key.ToString());
18.                  journalEntries.Add(tmp);
19.             }

20.             public override string ToString()
21.             {
22.                  StringBuilder stringBuilder = new StringBuilder();
23.                  foreach (var item in journalEntries)
24.                  {
25.                       stringBuilder.AppendLine(" - " + item.ToString());
26.                  }
27.                  return stringBuilder.ToString();
28.             }
29.        }
30.   }
```

```
1.    //-----------------------------------------------------------------------------------------||
2.    // Файл JournalEntry.cs
3.    //-----------------------------------------------------------------------------------------||

4.    namespace LR4
5.    {
6.         internal class JournalEntry
7.         {
8.              public string NameCollection { get; set; }
9.              public EAction Action { get; set; }
10.             public string Property { get; set; }
11.             public string Key {  get; set; }

12.             public JournalEntry(string Name, EAction action, string prop, string key)
13.             {
14.                  NameCollection = Name;
15.                  Action = action;
16.                  Property = prop;
17.                  Key = key;
18.             }

19.             public override string ToString()
20.             {
21.                  return NameCollection + " " + Action.ToString() + " " + Property + " " +
                     Key;
22.             }
23.        }
24.   }
```

```
//-----------------------------------------------------------------------------------------||
// Файл StudentCollection.cs
//-----------------------------------------------------------------------------------------||

using LR2;
using System.Text;

namespace LR4
{
    /// <summary>
    ///
    /// </summary>
    /// <typeparam name="TKey"></typeparam>
    /// <param name="st"></param>
    /// <returns></returns>
    delegate TKey KeySelector<TKey>(Student st);

    internal class StudentCollection<TKey>
    {
        #region --- Переменные ---

        internal double MaxAvgRating {
            get {
                if (Dictionary.Count == 0)
                    return 0;
                else
                    return Dictionary.Max(kvp=>kvp.Value.AvgRating);
            }
        }

        /// <summary>
        /// Выполняет группировку элементов коллекции
        /// Dictionary<TKey, Student> в зависимости от формы обучения студента с
        /// помощью метода Group класса System.Linq.Enumerable.
        /// </summary>
        internal IEnumerable<IGrouping<Education, KeyValuePair<TKey, Student>>> group
        {
            get
            {
                return Dictionary.GroupBy((kvp) => { return kvp.Value.Education; });
            }
        }

        /// <summary>
        /// Кеш инициализированных студентов
        /// </summary>
        Dictionary<TKey, Student> Dictionary;

        /// <summary>
        /// Делегат формирования ключей словаря
        /// </summary>
        KeySelector<TKey> keySelector;

        /// <summary>
        /// Имя коллекции
        /// </summary>
        string NameCollection;

        #endregion
```

```
        #region --- Events and Delegates ---

        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="TKey"></typeparam>
        /// <param name="source"></param>
        /// <param name="args"></param>
        public delegate void StudentsChangedHandler<TKey>(object source,
StudentsChangedEventArgs<TKey> args);

        /// <summary>
        ///
        /// </summary>
        public event StudentsChangedHandler<TKey> StudentsChanged;

        #endregion

        public StudentCollection(string name, KeySelector<TKey> deleg)
        {
            keySelector = deleg;
            NameCollection = name;
            Dictionary = new Dictionary<TKey, Student>();
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        /// <exception cref="NotImplementedException"></exception>
        void Student_PropertyChanged(object? sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            var key = Dictionary.FirstOrDefault(x => x.Value == sender as Student).Key;
            if (key != null)
                StudentsChanged(this, new StudentsChangedEventArgs<TKey>(NameCollection,
EAction.Property, e.PropertyName, key));
        }


        /// <summary>
        /// Возвращает подмножество элементов коллекции
        /// Dictionary<TKey, Student> с заданной формой обучения;
        /// для формирования подмножества использовать метод Where класса
System.Linq.Enumerable;
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public IEnumerable<KeyValuePair<TKey, Student>> EducationForm(Education value)
        {
            return Dictionary.Where((kvp)=>kvp.Value.Education == value);
        }
```

```
        /// <summary>
        /// Для добавления некоторого числа элементов типа Student для инициализации коллекции
        /// по умолчанию;
        /// </summary>
        public void AddDefaults()
        {
            for (int i = 0; i < 1; i++) {
                var st = new Student();
                var key = keySelector.Invoke(st);
                if(Dictionary.TryAdd(key, st))
                {
                    StudentsChanged(this, new StudentsChangedEventArgs<TKey>(NameCollection,
EAction.Add, "", key));
                }

                st.PropertyChanged += Student_PropertyChanged;
            }
        }

        /// <summary>
        /// Для добавления элементов в коллекцию Dictionary<TKey, Student>
        /// </summary>
        /// <param name="students"></param>
        public void AddStudents(params Student[] students)
        {
            foreach (var student in students) {
                var key = keySelector.Invoke(student);
                if(Dictionary.TryAdd(key, student))
                {
                    StudentsChanged(this, new StudentsChangedEventArgs<TKey>(NameCollection,
EAction.Add, "", key));
                }

                student.PropertyChanged += Student_PropertyChanged;
            }
        }

        /// <summary>
        /// Для удаления элемента со значением st из словаря Dictionary<TKey, Student>;
        /// если в словаре нет элемента st, метод возвращает значение false;
        /// </summary>
        /// <param name="st"></param>
        /// <returns></returns>
        public bool Remove(Student st)
        {
            if (Dictionary.ContainsValue(st))
            {
                var key = Dictionary.FirstOrDefault(x => x.Value == st).Key;
                Dictionary.Remove(key);
                StudentsChanged(this, new StudentsChangedEventArgs<TKey>(NameCollection,
EAction.Remove, "", key));

                st.PropertyChanged -= Student_PropertyChanged;

                return true;
            }
            else
                return false;
        }
```

```
        /// <summary>
        /// Формирует строку, содержащую информацию обо всех элементах
        /// коллекции Dictionary<TKey, Student>, в том числе значения всех полей
        /// класса Student, включая список зачетов и экзаменов;
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            StringBuilder stringBuilder = new StringBuilder();

            foreach (var kvp in Dictionary)
            {
                stringBuilder.AppendLine("||----------------------------------------------------
------------||");
                stringBuilder.AppendLine($" {kvp.Key} - {kvp.Value.ToString()}");
                stringBuilder.AppendLine("||----------------------------------------------------
------------||");
            }

            return stringBuilder.ToString();
        }

        /// <summary>
        /// Формирует строку с информацией обо всех элементах коллекции Dictionary<TKey,
Student>, состоящую из
        /// значений всех полей, среднего балла, числа зачетов и экзаменов для
        /// каждого элемента Student, но без списка зачетов и экзаменов.
        /// </summary>
        /// <returns></returns>
        public virtual string ToShortString()
        {
            StringBuilder stringBuilder = new StringBuilder();

            foreach (var kvp in Dictionary)
            {
                stringBuilder.AppendLine("||----------------------------------------------------
------------||");
                stringBuilder.AppendLine($" {kvp.Key} - {kvp.Value.ToShortString()} " +
                    $"Exams: {kvp.Value.ClosedExams.Count()} Tests:
{kvp.Value.Subjects.Count()}");
                stringBuilder.AppendLine("||----------------------------------------------------
------------||");
            }

            return stringBuilder.ToString();
        }

    }
}
```

```
//-----------------------------------------------------------------------------------------------||
// Файл Student.cs
//-----------------------------------------------------------------------------------------------||

using System.Collections;
using System.ComponentModel;
using System.Text;

using LR2;
using LR3;
using Exam = LR3.Exam;

namespace LR4
{

    internal class Student : Person, IDateAndCopy, IEnumerable, INotifyPropertyChanged
    {
        #region --- Переменные ---
        Person info => this;

        /// <summary>
        /// Форма обучения
        /// </summary>
        Education education;

        /// <summary>
        /// Номер группы
        /// </summary>
        int numberGroup;

        /// <summary>
        /// Закрытые экзамены
        /// </summary>
        List<Exam> closedExams;

        /// <summary>
        ///
        /// </summary>
        List<Test> subjects;

        #endregion

        #region --- Свойства ---

        /// <summary>
        /// Средний балл
        /// </summary>
        internal double AvgRating {
            get {
                if (closedExams == null || closedExams.Count == 0) return 0;

                return (double)(closedExams.ToArray()?.Average((ex) => { return (ex as
    Exam)?.Rating ?? 0.0; }));
            }
        }
```

```
    internal Education Education
    {
        get { return education; }
        set {
            education = value;
            var args = new PropertyChangedEventArgs("education");
            PropertyChanged?.Invoke(this, args);
        }
    }
    internal int NumberGroup
    {
        get { return numberGroup; }
        set {
            //if (value <= 100 || value > 599)
            //    throw new Exception("Недопустимый номер группы, границы значений от 101 до
599");
            numberGroup = value;
            var args = new PropertyChangedEventArgs("numberGroup");
            PropertyChanged?.Invoke(this, args);
        }
    }
    internal List<Exam> ClosedExams
    {
        get { return closedExams; }
        set { closedExams = value; }
    }
    internal List<Test> Subjects
    {
        get { return subjects; }
        set { subjects = value; }
    }

    #endregion

    #region --- События и делегаты ---

    /// <summary>
    ///
    /// </summary>
    public event PropertyChangedEventHandler? PropertyChanged;

    #endregion

    #region --- Итераторы ---

    /// <summary>
    /// Итератор для последовательного перебора всех элементов (объектов типа object)
    /// из списков зачетов и экзаменов(объединение)
    /// </summary>
    /// <returns></returns>
    public IEnumerable<object> GetSubjects()
    {
        foreach (Exam ex in closedExams)
        {
            yield return ex;
        }

        foreach (Test subject in subjects)
        {
               yield return subject;
        }
    }
```

```
    /// <summary>
    /// Итератор с параметром для перебора экзаменов (объектов типа Exam)
    /// с оценкой больше заданного значения.
    /// </summary>
    /// <param name="targetRating"></param>
    /// <returns></returns>
    public IEnumerable<Exam> GetExam(int targetRating)
    {
        foreach (Exam ex in closedExams)
        {
            if (ex.Rating >= targetRating)
                yield return ex;
        }
    }

    /// <summary>
    /// По доп требованиям:
    /// определить итератор для перебора сданных зачетов и экзаменов
    /// (объектов типа object), для этого определить метод, содержащий блок
    /// итератора и использующий оператор yield; сданный экзамен - экзамен с
    /// оценкой больше 2;
    /// </summary>
    /// <param name="targetRating"></param>
    /// <returns></returns>
    public IEnumerable<object> GetPassedSubjectsAndExams()
    {
        var iterExam = closedExams.GetEnumerator();
        while (iterExam.MoveNext())
        {
            if ((iterExam.Current as Exam).Rating > 2)
                yield return iterExam.Current;
        }

        var iterTest = Subjects.GetEnumerator();
        while(iterTest.MoveNext())
        {
            if ((iterTest.Current as Test).isPassed)
                yield return iterTest.Current;
        }
    }

    /// <summary>
    /// По доп требованиям:
    /// определить итератор для перебора всех сданных зачетов (объектов
    /// типа Test), для которых сдан и экзамен, для этого определить метод,
    /// содержащий блок итератора и использующий оператор yield.
    /// </summary>
    /// <returns></returns>
    public IEnumerable<Test> GetPassedSubjects()
    {
        foreach (Exam exam in closedExams)
        {
            foreach (Test subject in Subjects)
            {
                if (subject.SubjectName == exam.NameSubject && exam.Rating > 2 &&
subject.isPassed)
                {
                    yield return subject;
                }
            }
        }
    }
```

```
/// <summary>
/// По доп. требованиям
/// </summary>
/// <returns></returns>
IEnumerator IEnumerable.GetEnumerator()
{
    return null;
    //return new StudentEnumerator(this);
}

#endregion

/// <summary>
///
/// </summary>
/// <param name="education"></param>
/// <returns></returns>
internal bool this[Education education]
{
    get => education == this.education;
}

#region --- Конструкторы ---

/// <summary>
///
/// </summary>
internal Student() {

    education = new ();
    numberGroup = 0;
    closedExams = new List<Exam>();
    Subjects = new List<Test>();
}

/// <summary>
///
/// </summary>
/// <param name="info"></param>
/// <param name="education"></param>
/// <param name="numberGroup"></param>
internal Student(Person person, Education education, int numberGroup )
{
    base._BirthData = person.BirthData;
    base._Surname = person.Surname;
    base._Forename = person.Forename;

    this.education = education;
    this.numberGroup = numberGroup;

    closedExams = new List<Exam>();
    Subjects = new List<Test>();
}

#endregion
```

```
    #region --- Virtuals ---

    public override string ToString()
    {
        StringBuilder exams = new StringBuilder();

        if (closedExams != null)
            foreach (var ex in ClosedExams)
            {
                exams.AppendLine(" - " + ex.ToString());
            }

        StringBuilder tests = new StringBuilder();

        if (subjects != null)
            foreach (var ex in subjects)
            {
                tests.AppendLine(" - " + ex.ToString());
            }

        return ToShortString() + "\n Exams:\n" + exams.ToString() + "\n Tests:\n" +
tests.ToString();
    }

    public virtual string ToShortString()
    {
        return base.ToString() + " " + education.ToString() + " " + numberGroup + ", AvgRat:
" + AvgRating;
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="obj"></param>
    /// <returns></returns>
    public override bool Equals(object obj)
    {
        Student tmp = obj as Student;
        return tmp != null &&
            base.Equals(tmp) &&
            this.education == tmp.education &&
            this.closedExams == tmp.closedExams &&
            this.AvgRating == tmp.AvgRating &&
            this.subjects == tmp.subjects &&
            this.numberGroup == tmp.numberGroup;
    }

    public static bool operator ==(Student p1, Student p2)
    {
        if (p1 is null || p2 is null)
            return false;

        return p1.Equals(p2);
    }
    public static bool operator !=(Student p1, Student p2)
    {
        return !(p1 == p2);
    }



    /// <summary>
    ///
```

```
        /// </summary>
        /// <returns></returns>
        public override int GetHashCode()
        {
            return info.GetHashCode() + education.GetHashCode() + numberGroup +
    closedExams.GetHashCode() + subjects.GetHashCode();
        }

        #endregion

        /// <summary>
        /// Добавляет перечень экзаменов в коллекцию закрытых экзаменов
        /// </summary>
        /// <param name="exams">Закрытые экзамены</param>
        internal void AddExams(List<Exam> exams)
        {
            if (closedExams != null)
            {
                if (exams != null)
                    closedExams.AddRange((exams).ToArray());
            }
            else
                closedExams = exams;
        }

        /// <summary>
        /// Нельзя прегрузить, ошибка CS0506
        /// </summary>
        /// <returns></returns>
        public new object DeepCopy()
        {
            var tmp = new Student((Person)info.DeepCopy(), education, numberGroup);
            tmp.closedExams = new List<Exam>();
            tmp.subjects = new List<Test>();

            foreach (Exam exam in closedExams)
            {
                tmp.closedExams.Add(exam.DeepCopy() as Exam);
            }

            foreach (Test subject in subjects)
            {
                tmp.subjects.Add(subject);
            }

            return tmp;
        }

    }
}
```