Лабораторная работа 4. Варианты второго уровня

Делегаты. События.

Требования к программе, общие для всех вариантов

В лабораторной работе требуется определить класс, содержащий типизированную коллекцию, который с помощью событий извещает об изменениях в коллекции и в данных ее элементов.

Коллекция состоит из объектов ссылочных типов. Набор ссылок, образующих коллекцию, изменяется при удалении/добавлении элементов или при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение.

Класс, содержащий коллекцию, сообщает об этих изменениях в коллекции с помощью событий, для которых определяется пользовательский делегат.

Данные элементов, входящих в коллекцию, изменяются при вызове метода get в свойствах, определенных в элементах коллекции. При этом значение самой ссылки, которая входит в коллекцию, не изменяется.

Элементы сообщают об изменениях в своих данных с помощью события PropertyChange интерфейса System.ComponentModel.INotifyPropertyChanged. Делегат, отвечающий этому событию,

Delegate void PropertyChangedEventHandler

(Objectsender, PropertyChangedEventArgs

определен в базовой библиотеке. В классе PropertyChangedEventArgs есть конструктор с единственным параметром типа string, через который обработчикам события обычно передается имя свойства, которое является источником изменений данных объекта.

Класс, содержащий коллекцию, должен подписаться на событие PropertyChanged для каждого элемента коллекции. Подписку на событие PropertyChanged надо выполнить во всех методах, которые добавляют в коллекцию новые элементы. При удалении элемента из коллекции от подписки на его событие PropertyChanged необходимо отказаться.

При подписке на событие PropertyChanged как обработчик события надо использовать метод класса с коллекцией. В этом методе события PropertyChanged, которые происходят в элементах коллекции, преобразуются в события класса, содержащего коллекцию. Для этого из информации, которую получает обработчик события PropertyChanged, формируются данные для нового события, которое бросается из класса-коллекции.

События регистрируются в специальных классах-слушателях. Класс слушатель должен отслеживать изменения только в тех объектах, которые входят в коллекцию.

Вариант 1.

Требования к программе

Определить новые версии классов Student и StudentCollection 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, определить делегат StudentsChangedHandler с сигнатурой:

void StudentsChangedHandler (object source, StudentsChangedEventArgs args);

Класс StudentsChangedEventArgs, 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 c параметрами типа string, Action, string и TKey для инициализации значений всех свойств класса;

• перегруженную версию метода string ToString().

В новую версию класса StudentCollection добавить

• открытое автореализуемое свойство типа string с названием коллекции; • метод bool Remove(Student st) для удаления элемента со значением st из словаря Dictionary; если в словаре нет элемента st, метод возвращает значение false;

• событие StudentsChanged типа StudentsChangedHandler, 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. 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 c 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;

• обработчик события StudentsChanged, который на основе информации из объекта StudentsChangedEventArgs, создает элемент JournalEntry и добавляет его в список List;

• перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List.

В методе Main()

1. Создать две коллекции StudentCollection с разными названиями.

2. Создать объект Journal и подписать его на события StudentsChanged из обеих коллекций StudentCollection.

3. Внести изменения в коллекции StudentCollection:

• 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

Код программы (описание в комментариях):

1. Ниже преведен класс Program и метод Main откуда начинается программа.

public class Program

{

static void Main(string[] args)

{

//Создание двух коллекций студентов

var collection1 = new StudentCollection<string>(KeySelect);

var collection2 = new StudentCollection<string>(KeySelect);

//Присваивание имен коллекциям

collection1.CollectionName = "TestListCollection1";

collection2.CollectionName = "TestListCollection2";

//Создание журнала

var journal1 = new Journal();

//Подписки на события обновления списков

collection1.StudentsChanged += journal1.StudentsCountChanged;

collection2.StudentsChanged += journal1.StudentsCountChanged;

//Добавление студентов

collection1.AddStudents(new Student(), new Student());

collection2.AddStudents(new Student(), new Student());

//Обновление существующего

collection1[0].Value.GroupNumber = 200;

collection2[0].Value.Education =

Practice.Models.Enums.EducationEnum.SecondEducation;

//Удаленпие

collection1.Remove(collection1.First());

collection2.Remove(collection2.First());

///Вывод на экран

Console.WriteLine(journal1.ToString());

}

/// <summary>

/// Создание ключа для студента

/// </summary>

/// <param name="student"></param>

/// <returns></returns>

public static string KeySelect(Student student)

{

return $"{student.Name} {student.LastName}";

}

}

1. Воссоздаем класс Person из прежних лабораторных работ

/// <summary>

/// Класс человека

/// </summary>

public class Person : IDateAndCopy, IComparer<Person>, IComparable

{

/// <summary>

/// Имя

/// </summary>

protected string \_name;

/// <summary>

/// Фамилия

/// </summary>

protected string \_lastName;

/// <summary>

/// Имя

/// </summary>

public string Name

{

get => \_name;

set => \_name = value;

}

/// <summary>

/// Фамилия

/// </summary>

public string LastName

{

get => \_lastName;

set => \_lastName = value;

}

/// <summary>

/// Год рождения

/// </summary>

public int CustomizeDateOfBirthYear

{

get

{

return Date.Year;

}

set

{

Date = Date.AddYears(-Date.Year).AddYears(value);

}

}

/// <summary>

/// Дата рождения

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Конструктор заполняющий дефолтными значениями

/// </summary>

public Person()

{

\_name = ProgramConsts.DefaultName;

\_lastName = StringExtension.GetRandom(10);

Date = DateTime.Now.AddYears(-(new Random().Next(1, 20)));

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

public Person(string name, string lastName, DateTime datetime)

{

\_name = name;

\_lastName = lastName;

Date = datetime;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}, Дата рождения {Date}";

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}";

}

/// <summary>

/// Сравнение оьъектов

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public override bool Equals(object obj)

{

if (Object.ReferenceEquals(this, obj))

return true;

if (this.GetType() != obj.GetType())

return false;

var person = (Person?)obj;

return this.Name == person.Name && LastName == person.LastName && Date

== person.Date;

}

/// <summary>

/// Проверка на равенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator == (Person person1, Person person2)

{

if(person1 is null)

{

if(person2 is null)

{

return true;

}

return false;

}

return person1.Equals(person2);

}

/// <summary>

/// Проверка на неравенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator != (Person person1, Person person2)

{

return !(person1 == person2);

}

/// <summary>

/// Получение хэш кода

/// </summary>

/// <returns></returns>

public override int GetHashCode()

{

return (Name, LastName, Date).GetHashCode();

}

/// <summary>

/// Создание полной копии

/// </summary>

/// <returns></returns>

public virtual object DeepCopy()

{

return new Person(this.Name, this.LastName, this.Date);

}

/// <summary>

/// Сравнение даты

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

/// <returns></returns>

/// <exception cref="Exception"></exception>

public int Compare(Person? x, Person? y)

{

if(x is null || y is null)

throw new Exception("Объект равен null");

if (x.Date > y.Date)

{

return 1;

}

else if (x.Date < y.Date)

return -1;

else

return 0;

}

/// <summary>

/// Сравнение объекта по фамилии

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

/// <exception cref="Exception"></exception>

public int CompareTo(object? obj)

{

if (obj is Person person)

return LastName.CompareTo(person.LastName);

throw new Exception("Объект не является типом " + nameof(Person));

}

}

1. Определяем новую версию класса Student

public class Student : Person, IDateAndCopy, INotifyPropertyChanged

{

/// <summary>

/// Образование

/// </summary>

private EducationEnum \_education;

/// <summary>

/// Номер группы

/// </summary>

private int \_groupNumber;

/// <summary>

/// Экзамены

/// </summary>

private List<Exam> \_exams;

/// <summary>

/// Зачеты

/// </summary>

private List<Test> \_tests;

/// <summary>

/// Событие изменения свойства

/// </summary>

public event PropertyChangedEventHandler? PropertyChanged;

/// <summary>

/// Образование

/// </summary>

public EducationEnum Education

{

get => \_education;

set

{

\_education = value;

if (PropertyChanged != null)

PropertyChanged.Invoke(this, new

PropertyChangedEventArgs(nameof(Education)));

}

}

/// <summary>

/// Номер группы

/// </summary>

public int GroupNumber

{

get

{

return \_groupNumber;

}

set

{

if (value <= 100 || value > 599)

throw new Exception("Значение не соответствует условию > 100 и

<= 599");

\_groupNumber = value;

if(PropertyChanged != null)

PropertyChanged.Invoke(this, new

PropertyChangedEventArgs(nameof(GroupNumber)));

}

}

/// <summary>

/// Экзамены

/// </summary>

public List<Exam> Exams

{

get => \_exams;

set => \_exams = value;

}

/// <summary>

/// Тесты

/// </summary>

public List<Test> Tests

{

get => \_tests;

set => \_tests = value;

}

/// <summary>

/// Средняя оценка

/// </summary>

public double AvgGrade { get => \_exams == null ? 0 : \_exams.ToArray().Average(c => ((Exam)c).Grade); }

/// <summary>

/// Дата рождения

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Индексатор

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public bool this[EducationEnum education]

{

get => \_education == education ? true : false;

}

public Person Person

{

get

{

return (Person)base.MemberwiseClone();

}

set

{

this.Name = value.Name;

this.LastName = value.LastName;

this.Date = value.Date;

}

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

/// <param name="personInfo"></param>

/// <param name="education"></param>

/// <param name="groupNumber"></param>

public Student(Person person, EducationEnum education, int groupNumber) :

base(person.Name, person.LastName, person.Date)

{

\_education = education;

\_groupNumber = groupNumber;

}

/// <summary>

/// Конструктор без параметров

/// </summary>

public Student() : base()

{

\_groupNumber = new Random().Next(200, 401);

\_education = (EducationEnum)new Random().Next(0,3);

}

/// <summary>

/// Добавление экзаменов

/// </summary>

/// <param name="exams"></param>

public void AddExams(List<Exam> exams)

{

if (\_exams == null)

\_exams = exams;

else

\_exams.AddRange(exams);

}

/// <summary>

/// Добавление тестов

/// </summary>

/// <param name="tests"></param>

public void AddTests(List<Test> tests)

{

if (\_tests == null)

\_tests = tests;

else

\_tests.AddRange(tests);

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

var str = $"\n{base.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber} Средний балл: {AvgGrade} \n";

if (\_exams != null && \_exams.Count > 0)

str += string.Join(",\n", \_exams.ToArray().Select(c =>

c.ToString()).ToList());

if(\_tests != null && \_tests.Count > 0)

str += "\n" + string.Join(",\n", \_tests.ToArray().Select(c =>

c.ToString()).ToList());

return str;

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"{base.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber}, Средний балл: {AvgGrade}, Кол-во экзаменов:

{\_exams.Count}, Кол-во зачетов: {\_tests.Count}\n";

}

/// <summary>

/// Получение полной копии

/// </summary>

/// <returns></returns>

public override object DeepCopy()

{

var newStudent = new Student((Person)base.DeepCopy(), Education,

GroupNumber);

var newExams = new List<Exam>();

var newTests = new List<Test>();

foreach(var item in Exams)

{

newExams.Add((Exam)item.DeepCopy());

}

foreach(var item in Tests)

{

newTests.Add((Test)item.DeepCopy());

}

newStudent.AddExams(newExams);

newStudent.AddTests(newTests);

return newStudent;

}

/// <summary>

/// Итератор

/// </summary>

/// <returns></returns>

public IEnumerable GetEnumerator(double? minGrade)

{

if (minGrade == null)

{

foreach (var item in Exams)

{

yield return item;

}

foreach(var item in Tests)

{

yield return item;

}

}

else

foreach (var item in Exams)

{

if (((Exam)item).Grade > minGrade)

yield return (Exam)item;

}

}

}

1. Воссоздаем класс Exam из прежних лабораторных работ

/// <summary>

/// Экзамен

/// </summary>

public class Exam : IDateAndCopy

{

/// <summary>

/// Предмет

/// </summary>

public string Subject { get; set; }

/// <summary>

/// Оценка

/// </summary>

public int Grade { get; set; }

/// <summary>

/// Начало экзамена

/// </summary>

public DateTime DateStart { get; set; }

/// <summary>

/// Дата

/// </summary>

public DateTime Date { get; set; }

public Exam()

{

Subject = StringExtension.GetRandom(7);

Grade = new Random().Next(1, 6);

DateStart = DateTime.UtcNow;

}

public Exam(string subject, int grade, DateTime date)

{

Subject = subject;

Grade = grade;

DateStart = date;

}

/// <summary>

/// Получение строки со всеми полями

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Предмет: {Subject}, Оценка: {Grade}, Дата начала: {DateStart}";

}

/// <summary>

/// Полуение полнй копии

/// </summary>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

public object DeepCopy()

{

return new Exam(Subject, Grade, DateStart);

}

}

1. Воссоздаем класс Test из прежних лабораторных работ

public class Test : IDateAndCopy

{

/// <summary>

/// Название

/// </summary>

public string Name { get; set; }

/// <summary>

/// Сдан ли

/// </summary>

public bool IsPassed { get; set; }

/// <summary>

/// Дата сдачи

/// </summary>

public DateTime Date { get; set; }

public Test(string name, bool isPassed)

{

Name = name;

IsPassed = isPassed;

}

public Test()

{

Name = StringExtension.GetRandom(10);

IsPassed = false;

}

/// <summary>

/// Преобразование объекта к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Имя: {Name}, Зачет: {IsPassed}";

}

/// <summary>

/// Полное копирование

/// </summary>

/// <returns></returns>

public object DeepCopy()

{

return new Test(Name, IsPassed);

}

}

1. Определяем новую версию класса StudentCollection для описания коллекции студентов.

public class StudentCollection<TKey> : IEnumerable<Student>

{

/// <summary>

/// Студенты

/// </summary>

private Dictionary<TKey, Student> \_students = new Dictionary<TKey,

Student>();

/// <summary>

/// Делегат для изменения коллекции

/// </summary>

/// <typeparam name="TKey"></typeparam>

/// <param name="sourse"></param>

/// <param name="args"></param>

public delegate void StudentsChangedHandler<TKey>(object sourse,

StudentsChangedEventArgs<TKey> args);

/// <summary>

/// Событие для изменений коллекции

/// </summary>

public event StudentsChangedHandler<TKey> StudentsChanged;

/// <summary>

/// Делегат для селектора

/// </summary>

/// <typeparam name="TKey"></typeparam>

/// <param name="st"></param>

/// <returns></returns>

public delegate TKey KeySelector<TKey>(Student st);

/// <summary>

/// Селектор ключа

/// </summary>

private readonly KeySelector<TKey> \_keySelector;

public StudentCollection(KeySelector<TKey> selector)

{

\_keySelector = selector;

}

/// <summary>

/// Название коллекции

/// </summary>

public string CollectionName { get; set; } = "ListCollection";

/// <summary>

/// Максимальная оценка

/// </summary>

public double MaxGrade

{

get

{

if (\_students == null || \_students.Count == 0)

return 0;

return \_students.Max(c=> c.Value.AvgGrade);

}

}

/// <summary>

/// Получение специалистов

/// </summary>

/// <returns></returns>

public IEnumerable<Student> StudentsWithSpecialist

{

get

{

foreach (var student in \_students.Where(c => c.Value.Education ==

Practice.Models.Enums.EducationEnum.Specialist))

{

yield return student.Value;

}

}

}

/// <summary>

/// Получение студентов со средним баллом

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

public List<Student> AverageMarkGroup(double value)

{

return \_students.GroupBy(c => c.Value.AvgGrade).FirstOrDefault(c =>

c.Key == value)?.Select(c => c.Value).ToList() ?? new List<Student>();

}

/// <summary>

/// Добавление студентов

/// </summary>

public void AddStudents(params Student[] students)

{

foreach(var item in students)

{

item.PropertyChanged += Item\_PropertyChanged;

if (\_students.ContainsKey(\_keySelector(item)))

\_students[\_keySelector(item)] = item;

else

\_students.Add(\_keySelector(item), item);

if (StudentsChanged != null)

StudentsChanged.Invoke(this, new

StudentsChangedEventArgs<TKey>(CollectionName,

Practice4\_2.Models.Enums.ActionEnum.Add, nameof(\_students),

\_keySelector(item)));

}

}

/// <summary>

/// Обработчик события изменения свойств студента

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

/// <exception cref="NotImplementedException"></exception>

private void Item\_PropertyChanged(object? sender, PropertyChangedEventArgs e)

{

StudentsChanged.Invoke(sender, new

StudentsChangedEventArgs<TKey>(CollectionName,

Practice4\_2.Models.Enums.ActionEnum.Property, e.PropertyName,

\_keySelector((Student)sender)));

}

/// <summary>

/// Добавление дефолтных значений

/// </summary>

public void AddDefaults(int count)

{

for(int i = 0; i < count; i++)

{

var stud = new Student();

stud.PropertyChanged += Item\_PropertyChanged;

stud.Exams = new List<Exam>();

for(int j = 0; j < 5; j++)

{

stud.Exams.Add(new Exam());

}

\_students.Add(\_keySelector(stud), stud);

if (StudentsChanged != null)

StudentsChanged.Invoke(this, new

StudentsChangedEventArgs<TKey>(CollectionName,

Practice4\_2.Models.Enums.ActionEnum.Add, nameof(\_students),

\_keySelector(stud)));

}

}

public override string ToString()

{

return String.Join("", \_students.Select(c => c.Value.ToString()));

}

/// <summary>

/// Возвращает короткую версию строки элемента

/// </summary>

/// <returns></returns>

public string ToShortString()

{

return String.Join("\n\n", \_students.Select(c =>

c.Value.ToShortString()));

}

/// <summary>

/// Получение перечисления

/// </summary>

/// <returns></returns>

public IEnumerator<Student> GetEnumerator()

{

return \_students.Values.GetEnumerator();

}

/// <summary>

/// Получение перечисления

/// </summary>

/// <returns></returns>

IEnumerator IEnumerable.GetEnumerator()

{

return \_students.GetEnumerator();

}

/// <summary>

/// Метод удаления элемента

/// </summary>

/// <param name="j"></param>

/// <returns></returns>

public bool Remove(Student st)

{

try

{

var key = \_students.FirstOrDefault(c => c.Value == st);

if (key.Key != null)

{

if (StudentsChanged != null)

{

StudentsChanged.Invoke(this, new

StudentsChangedEventArgs<TKey>(CollectionName, Practice4\_2.Models.Enums.ActionEnum.Remove, nameof(\_students), key.Key));

key.Value.PropertyChanged -= Item\_PropertyChanged;

}

\_students.Remove(key.Key);

}

else

return false;

return true;

}

catch(Exception ex)

{

return false;

}

}

/// <summary>

/// индексатор

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public KeyValuePair<TKey, Student> this[int index]

{

get => \_students.ElementAt(index);

set

{

if (\_students.ContainsKey(value.Key))

\_students[value.Key] = value.Value;

else

\_students.Add(value.Key, value.Value);

if(StudentsChanged != null)

StudentsChanged.Invoke(this, new

StudentsChangedEventArgs<TKey>(CollectionName,

Practice4\_2.Models.Enums.ActionEnum.Add, nameof(\_students),

value.Key));

}

}

}

1. Определяем класс StudentChangedEventArgs для описания модели передаваемой в обработчик события

public class StudentsChangedEventArgs<TKey> : EventArgs

{

/// <summary>

/// Название теста

/// </summary>

public string CollectionName { get; set; }

/// <summary>

/// Название операции проводимой над коллекций

/// </summary>

public ActionEnum OperationName { get; set; }

/// <summary>

/// Объект студента

/// </summary>

public string Property { get; set; }

/// <summary>

/// Ключ

/// </summary>

public TKey Key { get; set; }

public StudentsChangedEventArgs(string collectionName, ActionEnum action,

string prop, TKey key)

{

Property = prop;

CollectionName = collectionName;

OperationName = action;

Key = key;

}

public StudentsChangedEventArgs() { }

/// <summary>

/// Получение строки с объекта

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Название коллекции: {CollectionName}\nТип операции:

{OperationName}\nСвойство: {Property}\nКлюч: {Key}";

}

}

1. Создаем класс Journal для описания журнала событий

public class Journal

{

/// <summary>

/// Список записей журнала

/// </summary>

private List<JournalEntry> \_journalsEntry;

public Journal()

{

\_journalsEntry = new List<JournalEntry>();

}

/// <summary>

/// Обработчик обновления списка студентов

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public void StudentsCountChanged(object sourse,

StudentsChangedEventArgs<string> args)

{

\_journalsEntry.Add(new JournalEntry

{

CollectionName = args.CollectionName,

OperationName = args.OperationName,

Property = args.Property,

Key = args.Key,

});

}

/// <summary>

/// Конверт к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Join("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", \_journalsEntry.Select(c =>

c.ToString()));

}

}

1. Создаем класс JournalEntry для описания элемента журнала событий

/// <summary>

/// Журнал операций

/// </summary>

public class JournalEntry

{

/// <summary>

/// Название коллекции запись которой создана

/// </summary>

public string CollectionName { get; set; }

/// <summary>

/// Название операции

/// </summary>

public ActionEnum OperationName { get; set; }

/// <summary>

/// Информация о студенте

/// </summary>

public string Property { get; set; }

/// <summary>

/// Ключ

/// </summary>

public string Key { get; set; }

public JournalEntry(string collectionName, ActionEnum operationName, string

student)

{

CollectionName = collectionName;

OperationName = operationName;

Property = student;

}

public JournalEntry() { }

/// <summary>

/// Преобразование к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Название коллекции: {CollectionName}\nНазвание операции:

{OperationName}\n:Свойство: {Property}\nКлюч: {Key} ";

}

}

Консольный вывод:

