Задание:

Вариант 3. Требования к программе

Определить новые версии классов **ResearchTeam** и **ResearchTeamCollection<TKey>**  из лабораторной работы 3.

Новая версия класса **ResearchTeam** реализует интерфейс **System.ComponentModel.INotifyPropertyChanged**. Событие **PropertyChanged** из интерфейса System.ComponentModel.INotifyPropertyChanged происходит при изменении значений свойств класса **ResearchTeam**, связанных с названием темы и продолжительностью исследований. 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) **Revision** со значениями **Remove, Replace** и **Property**.

Для события **ResearchTeamsChanged**, которое бросают методы новой версии класса ResearchTeamCollection<TKey>, определить универсальный делегат **ResearchTeamsChangedHandler<TKey>** с сигнатурой:

void ResearchTeamsChangedHandler<TKey>

(object source, ResearchTeamsChangedEventArgs<TKey> args);

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

* открытое автореализуемое свойство типа string с названием коллекции;
* открытое автореализуемое свойство типа Revision с информацией о том, чем вызвано событие, – удалением, заменой элемента или изменением данных элемента;
* открытое автореализуемое свойство типа string с названием свойства класса ResearchTeam, которое является источником изменения данных элемента; для событий, брошенных при удалении или замене элемента, значение свойства – пустая строка;
* открытое автореализуемое свойство типа int c номером регистрации объекта ResearchTeam для элемента, который был удален, в том числе и при замене элемента, или элемента, данные которого были изменены;
* конструктор c параметрами типа string, Revision, string и int для инициализации значений всех свойств класса;
* перегруженную версию метода string ToString().

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

* открытое автореализуемое свойство типа string с названием коллекции;
* метод bool Remove(ResearchTeam rt) для удаления элемента со значением rt из словаря Dictionary<TKey, ResearchTeam>; если в словаре нет элемента rt, метод возвращает значение false;
* метод bool Replace(ResearchTeam rtold, ResearchTeam rtnew) для замены в словаре Dictionary<TKey, ResearchTeam > элемента со значением rtold на элемент со значением rtnew; если в словаре нет элемента со значением rtold, метод возвращает значение false;
* событие **ResearchTeamsChanged** типа ResearchTeamsChangedHandler<TKey>,которое происходит, когдаизменяется набор элементов в коллекции-словаре Dictionary<TKey,ResearchTeam> или изменяются данные одного из ее элементов.

Определить класс **TeamsJournal**, который можно использовать для накопления информации об изменениях в коллекциях типа ResearchTeamCollection<TKey>. Класс TeamsJournal содержит список из элементов типа **TeamsJournalEntry**. Каждыйэлемент TeamsJournalEntry содержит информацию об отдельном изменении объекта ResearchTeamCollection<TKey>, в результате которого произошло событие ResearchTeamsChanged.

Класс **TeamsJournalEntry** содержит автореализуемые свойства

* типа string с названием коллекции;
* типа Revision c информацией о типе события;
* типа string с названием свойства класса ResearchTeam, которое явилось причиной изменения данных элемента;
* типа int с номером регистрации объекта ResearchTeam для удаленного элемента или элемента, данные которого были изменены;
* конструктор для инициализации всех свойств класса;
* перегруженную версию метода string ToString().

Класс **TeamsJournal** содержит

* закрытое поле List<TeamsJournalEntry> для списка изменений;
* обработчик события ResearchTeamsChanged; обработчик использует информацию, которая передается ему через объект ResearchTeamsChangedEventArgs, создает элемент TeamJournalEntry и добавляет его к списку List<TeamsJournalEntry>;
* перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<TeamsJournalEntry>.

В методе **Main()**

1. Создать две коллекции ResearchTeamCollection<string>.
2. Создать объект TeamsJournal, подписать его на события ResearchTeamsChanged из обоих объектов ResearchTeamCollection<string>.
3. Внести изменения в коллекции ResearchTeamCollection<string>

* добавить элементы в коллекции;
* изменить значения разных свойств элементов, входящих в коллекцию;
* удалить элемент из коллекции;
* изменить данные в удаленном элементе;
* заменить один из элементов коллекции;
* изменить данные в элементе, который был удален из коллекции при замене элемента.

1. Вывести данные объекта TeamsJournal.

Код:

**ResearchTeamCollection**

class ResearchTeamCollection<TKey> : IEnumerable

{

private List<ResearchTeam> SomeResearchTeams = new List<ResearchTeam>();// акрытое поле типа System.Collections.Generic.List<ResearchTeam>;

//открытое автореализуемое свойство типа string с названием коллекции

public delegate void TeamListHandler(object source, TeamListHandlerEventArgs args);

public ResearchTeamCollection(string name)

{

CollectionName = name;

SomeResearchTeams = new List<ResearchTeam>();

}

//открытое автореализуемое свойство типа string с названием коллекции;

public string CollectionName { get; set; }

//метод bool Remove(ResearchTeam rt) для удаления элемента со значением rt из словаря Dictionary<TKey, ResearchTeam>; если в словаре нет элемента rt, метод возвращает значение false

public bool Remove(ResearchTeam rt)

{

if (!SomeResearchTeams.Contains(rt))

return false;

SomeResearchTeams.Remove(rt);

ResearchTeamChanged?.Invoke(this, new ResearchTeamChangedEventArgs("ResearchTeamCollection", Revision.Remove, "ResearchTeam", rt.RegistrationNumber));

return true;

}

//метод bool Replace(ResearchTeam rtold, ResearchTeam rtnew) для замены в словаре Dictionary<TKey, ResearchTeam > элемента со значением rtold на элемент со значением rtnew; если в словаре нет элемента со значением rtold, метод возвращает значение false

public bool Replace(ResearchTeam rtold, ResearchTeam rtnew)

{

if (!SomeResearchTeams.Contains(rtold))

return false;

SomeResearchTeams[SomeResearchTeams.IndexOf(rtold)] = rtnew;

ResearchTeamChanged?.Invoke(this, new ResearchTeamChangedEventArgs("ResearchTeamCollection", Revision.Replace, "ResearchTeam", rtnew.RegistrationNumber));

return true;

}

//событие ResearchTeamsChanged типа ResearchTeamsChangedHandler<TKey>, которое происходит, когда изменяется набор элементов в коллекции-словаре Dictionary<TKey,ResearchTeam> или изменяются данные одного из ее элементов

public event ResearchTeamChangedHandler? ResearchTeamChanged;

//метод void InsertAt (int j, ResearchTeam rt), который вставляет элемент rt в список List<ResearchTeam> перед элементом с номером j; если в списке нет элемента с номером j, метод добавляет элемент в конец списка

public void InsertAt(int j, ResearchTeam rt)

{

if (SomeResearchTeams.Count < j)

{

SomeResearchTeams[SomeResearchTeams.Count - 1] = rt;

ResearchTeamAdded(SomeResearchTeams[SomeResearchTeams.Count - 1], new TeamListHandlerEventArgs("Коллекция в InsertAt", "Last element added", SomeResearchTeams.Count - 1));

}

for (int i = 0; i < SomeResearchTeams.Count; i++)

{

if (i - 1 == j)

{

SomeResearchTeams.Insert(i, rt);

ResearchTeamInserted(SomeResearchTeams[i], new TeamListHandlerEventArgs("Коллекция в InsertAt", "Element was added", j));

}

}

}

//индексатор типа ResearchTeam (с методами get и set) с целочисленным индексом для доступа к элементу списка List<ResearchTeam> с заданным номером

public ResearchTeam this[int index]

{

get { return SomeResearchTeams[index]; }

set { SomeResearchTeams[index] = value; }

}

public event TeamListHandler ResearchTeamAdded;

public event TeamListHandler ResearchTeamInserted;

public void AddDefaults()

{

ResearchTeamAdded?.Invoke(this, new TeamListHandlerEventArgs("InsertEvent", "Item added as last (from AddDefaults func)", SomeResearchTeams.Count));

SomeResearchTeams.Add(new ResearchTeam());

SomeResearchTeams.Add(new ResearchTeam());

SomeResearchTeams.Add(new ResearchTeam());

}

public void AddResearchTeams(params ResearchTeam[] rts)

{

ResearchTeamAdded?.Invoke(this, new TeamListHandlerEventArgs("InsertEvent", "Item added as last (from AddResearchTeams func)", SomeResearchTeams.Count));

SomeResearchTeams.AddRange(rts);

}

public int minNum

{

get

{

if (SomeResearchTeams.Count == 0) return -1;

return SomeResearchTeams.Min((e) => e.RegistrationNumber);

}

}

//перегруженную версию виртуального метода string ToString() для формирования строки c информацией обо всех элементах списка List<ResearchTeam>, которая содержит значения всех полей, список

public override string ToString()

{

string ResTeamString = "";

foreach (ResearchTeam team in SomeResearchTeams)

{

ResTeamString += team.ToString();

}

return ResTeamString;

}

//по номеру регистрации с использованием интерфейса IComparable, реализованного в классе Team;

public void ToSortByRegistrNumber()

{

SomeResearchTeams.Sort((x, y) => x.RegistrationNumber.CompareTo(y.RegistrationNumber));

}

//по названию темы исследований с использованием интерфейса IComparer<ResearchTeam>, реализованного в классе ResearchTeam;

public void SortByString()

{

SomeResearchTeams.Sort();

}

//по числу публикаций с использованием интерфейса IComparer<ResearchTeam>, реализованного во вспомогательном классе.

public void SortByPublications()

{

ResearchTeamComparer comp = new ResearchTeamComparer();

SomeResearchTeams.Sort(comp);

}

// свойство типа int (только с методом get), возвращающее минимальное значение номера регистрации для элементов списка List<ResearchTeam>;

public int MinRegNumber

{

get

{

if (SomeResearchTeams.Count == 0)

{

return 0;

}

return SomeResearchTeams.Min(teams => teams.RegistrationNumber);

}

}

//свойство типа IEnumerable<ResearchTeam> (только с методом get), возвращающее подмножество элементов списка List<ResearchTeam> с продолжительностью исследований TimeFrame.TwoYears; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

public IEnumerable<ResearchTeam> TwoYearsLong

{

get

{

IEnumerable<ResearchTeam> TwoTearsL = SomeResearchTeams.Where(time => time.ResearchDuration == TimeFrame.TwoYears);

return TwoTearsL;

}

}

//• метод List<ResearchTeam> NGroup(int value), который возвращает список, в который входят элементы ResearchTeam из списка List<ResearchTeam> с заданным числом участников исследования; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

public IEnumerator GetEnumerator()

{

for (int i = 0; i < SomeResearchTeams.Count; i++)

{

yield return SomeResearchTeams[i];

}

}

public event PropertyChangedEventHandler? PropertyChanged;

public delegate void ResearchTeamChangedHandler(object source, ResearchTeamChangedEventArgs args);

}

}

**TeamListHandlerEventArgs**

class TeamListHandlerEventArgs : EventArgs

{

//открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие

public string EventedCollectionName { get; set; }

//открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции

public string EventInfo { get; set; }

//открытое автореализуемое свойство типа int с номером элемента, который был добавлен или заменен

public int ElementNumber { get; set; }

//• конструкторы для инициализации класса

public TeamListHandlerEventArgs(string eventedCollectionName, string eventInfo, int elementNumber)

{

EventedCollectionName = eventedCollectionName;

EventInfo = eventInfo;

ElementNumber = elementNumber;

}

//• конструкторы для инициализации класса

public TeamListHandlerEventArgs() : this("SomCollection", "Some changes", 0) { }

//перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса

public override string ToString()

{

return $"Name of collection that throw event: {EventedCollectionName}\nEvent Info:\n\t{EventInfo}\nNumber of collection element, where event was thrown: {ElementNumber}";

}

}

**TeamsJournal**

class TeamsJournal

{

//закрытое поле List<TeamsJournalEntry> для списка изменений

private List<TeamsJournalEntry> ListOfChanges = new List<TeamsJournalEntry>();

public void TeamEventHandler(object source, TeamListHandlerEventArgs args)

{

ListOfChanges.Add(new TeamsJournalEntry(args.EventedCollectionName, "ReserchTeam was changed", args.EventInfo, args.ElementNumber));

}

//обработчик события ResearchTeamsChanged; обработчик использует информацию, которая передается ему через объект ResearchTeamsChangedEventArgs, создает элемент TeamJournalEntry и добавляет его к списку List<TeamsJournalEntry>

public void ResearchTeamChanged(object source, ResearchTeamChangedEventArgs args)

{

ListOfChanges.Add(new TeamsJournalEntry(args.NameCollection, "ReserchTeam was changed", args.NamePropertyResearchTeam, args.NumberRegisterResearchTeam));

}

public void ResearchTeamPropertyHandler(object source, PropertyChangedEventArgs args)

{

ListOfChanges.Add(new TeamsJournalEntry("Property was changed", args.PropertyName));

}

//перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<TeamsJournalEntry>

public override string ToString()

{

string str = "";

foreach (TeamsJournalEntry en in ListOfChanges)

{

str += en.ToString() + "\n";

}

return str;

}

}

**TeamsJournalEntry**

class TeamsJournalEntry

{

//типа string с названием коллекции

public string EventedCollectionName { get; set; }

//• типа string с названием свойства класса ResearchTeam, которое явилось причиной изменения данных элемента;

public string EventInformation { get; set; }

//типа int с номером регистрации объекта ResearchTeam для удаленного элемента или элемента, данные которого были изменены

public int ElementNumber { get; set; }

public string PropertyName { get; set; }

//конструктор для инициализации полей класса

public TeamsJournalEntry(string eventInformation, string propertyName)

{

EventInformation = eventInformation;

PropertyName = propertyName;

}

public TeamsJournalEntry(string collectionName, string eventInformation, string propertyName, int registrationNumber)

{

EventedCollectionName = collectionName;

EventInformation = eventInformation;

PropertyName = propertyName;

ElementNumber = registrationNumber;

}

//перегруженную версию метода string ToString()

public override string ToString()

{

return string.Format("Название коллекции: {0} \n Событие: {1} \n Кол-во элементов {2} \n", this.EventedCollectionName, this.EventInformation, this.ElementNumber);

}

}

}

**ResearchTeamChangedEventArgs**

class ResearchTeamChangedEventArgs : System.EventArgs

{

//типа string с названием коллекции;

public string NameCollection { get; set; }

//• типа Revision c информацией о типе события;

public Revision InfoAboutEvent { get; set; }

//типа string с названием свойства класса ResearchTeam, которое явилось причиной изменения данных элемента

public string NamePropertyResearchTeam { get; set; }

//типа int с номером регистрации объекта ResearchTeam для удаленного элемента или элемента, данные которого были изменены

public int NumberRegisterResearchTeam { get; set; }

public ResearchTeamChangedEventArgs(string str, Revision rev, string str1, int i)

{

NameCollection = str;

InfoAboutEvent = rev;

NamePropertyResearchTeam = str1;

NumberRegisterResearchTeam = i;

}

public override string ToString()

{

return String.Format("Название коллекции - {0},\n Информация, чем вызвано событие: {1}\n Название свойства класса archTeam, которое является источником изменения данных элемента: {2}\n Номер регистрации объекта ResearchTeam {3}", NameCollection, InfoAboutEvent, NamePropertyResearchTeam, NumberRegisterResearchTeam);

}

}

}

**ResearchTeam**

class ResearchTeam : Team, INameAndCopy, IComparable<ResearchTeam>, INotifyPropertyChanged//Определить класс ResearchTeam, который имеет

{

private string Theme; // закрытое поле типа string c названием темы исследований;

private string NameOfOrganisation; // закрытое поле типа string с названием организации;

private int RegNumber; //• закрытое поле типа int – регистрационный номер;

public TimeFrame ResearchDuration; //закрытое(уже нет) поле типа TimeFrame для информации о продолжительности исследований;

//закрытое поле типа System.Collections.ArrayList для списка публикаций (объектов типа Paper).

List<Person> ProjectParticipants = new List<Person>();

List<Paper> Publications = new List<Paper>();

public List<Paper> ListOfPublication { get { return Publications; } set { Publications = value; } }

Random rand = new Random();

public event PropertyChangedEventHandler? PropertyChanged;

public List<Person> ListOfParticipants { get { return ProjectParticipants; } set { ProjectParticipants = value; } }

//В классе ResearchTeam определить конструкторы:

public ResearchTeam( string InvestigationTheme, string Organisation, int RegistrationNumber, TimeFrame InvestigationDuration)// конструктор c параметрами типа string, Person, DateTime для инициализации всех свойств класса

{

ResearchDuration = InvestigationDuration;

Theme = InvestigationTheme;

RegNumber = RegistrationNumber;

NameOfOrganisation = Organisation;

}

// конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

public ResearchTeam() {

this.Theme = "Котики";

NameOfOrganisation = "КГЭУ";

ResearchDuration = TimeFrame.Year;

RegNumber = rand.Next(0, 100);

}

public override string ToString()

{

string stringListOfPublications = "";

foreach (Paper pap in Publications)

{

stringListOfPublications += pap.ToString() + "\r\n";

}

return string.Format("\r\n Тема: {0}, Длительность: {1} \r\n Организация:{2} \r\n Публикация: {3} \r\n Автор: {4}", Theme, ResearchDuration, NameOfOrganisation, stringListOfPublications, ProjectParticipants);

}

//виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка публикаций.

public string ToShortString()

{

return string.Format("\r\n Тема: {0}, Длительность: {1} \r\n Организация:{2} \r\n", Theme, ResearchDuration, NameOfOrganisation);

}

public object DeepCopy()

{

ResearchTeam other = (ResearchTeam)MemberwiseClone();

other.NameOfOrganisation = string.Copy(NameOfOrganisation);

other.ResearchDuration = ResearchDuration;

other.Theme = string.Copy(Theme);

other.RegNumber = RegNumber;

other.Publications = new List<Paper>(Publications);

other.ProjectParticipants = new List<Person>(ProjectParticipants);

return other;

}

//метод void AddMembers ( params Person[] ) для добавления элементов в список участников проекта;

public void AddMembers(params Person[] particips)

{

foreach (var item in particips)

{

ProjectParticipants.Add(item);

}

}

public Team Team //свойство типа Team; метод get свойства возвращает объект типа Team, данные которого совпадают с данными подобъекта базового класса, метод set присваивает значения полям из подобъекта базового класса;

{

get

{

return new Team(NameOfOrganisation, RegNumber);

}

set

{

Organisation = value.Organisation;

RegistrationNumber = value.RegistrationNumber;

}

}

//итератор для последовательного перебора участников проекта (объектов типа Person), не имеющих публикаций;

public IEnumerable<Person> MembersWithoutPublications()

{

ArrayList AutorsWithoutP = new ArrayList();

bool Bool;

foreach (Person pers in ProjectParticipants)

{

Bool = true;

foreach (Paper pap in Publications)

{

if (pap.Author == pers)

{

Bool = false;

break;

}

}

if (Bool)

{

AutorsWithoutP.Add(pers);

Console.WriteLine(pers.ToShortString());

}

}

for (int i = 0; i < AutorsWithoutP.Count; i++)

{

yield return (Person)AutorsWithoutP[i];

Console.Write(((Person)AutorsWithoutP[i]).ToShortString());

}

}

//итератор с параметром типа int для перебора публикаций, вышедших за последние n лет, в котором число n передается через параметр итератора.

public IEnumerable<Paper> LastPapers(int N\_years)

{

for (int i = 0; i < Publications.Count; i++)

{

if (((Paper)Publications[i]).Data.Year >= DateTime.Now.Year - N\_years)

{

yield return (Paper)Publications[i];

Console.Write(((Paper)Publications[i]).ToString());

}

}

}

public IEnumerator GetEnumerator()

{

foreach (Person person in ProjectParticipants)

{

yield return person;

}

}

public int CompareTo(ResearchTeam other)

{

if (other == null)

{

throw new ArgumentException("Wrong argument!");

}

return Theme.CompareTo(other.Theme);

}

delegate int KeySelector<TKey>(ResearchTeam rt);

public string name

{

get

{

return Theme;

}

set

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("name")); ; this.Theme = value;

}

}

public TimeFrame time

{

get

{

return ResearchDuration;

}

set

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("time")); this.ResearchDuration = value;

}

}

public enum Revision

{

Remove,

Replace,

Property

}

class ResearchTeamChangedEventArgs<TKey> : System.EventArgs

{

public string NameCollection { get; set; }

public Revision InfoAboutEvent { get; set; }

public string NamePropertyResearchTeam { get; set; }

public int NumberRegisterResearchTeam { get; set; }

public ResearchTeamChangedEventArgs(string str, Revision rev, string str1, int i)

{

NameCollection = str;

InfoAboutEvent = rev;

NamePropertyResearchTeam = str1;

NumberRegisterResearchTeam = i;

}

public override string ToString()

{

return String.Format("Название коллекции - {0},\n Информация, чем вызвано событие: {1}\n Название свойства класса archTeam, которое является источником изменения данных элемента: {2}\n Номер регистрации объекта ResearchTeam {3}", NameCollection, InfoAboutEvent, NamePropertyResearchTeam, NumberRegisterResearchTeam);

}

}

}

}

**Main**

static void Main(string[] args)

{

//Console.BackgroundColor = ConsoleColor.White;

//Console.ForegroundColor = ConsoleColor.Green;

//1. Создать две коллекции ResearchTeamCollection

ResearchTeamCollection<string> researchTeamString1 = new ResearchTeamCollection<string>("First Collection");

ResearchTeamCollection<string> researchTeamString2 = new ResearchTeamCollection<string>("2 Collection");

ResearchTeamCollection<string> researchTeamString = new ResearchTeamCollection<string>("2 Collection");

//2. Создать объект TeamsJournal

TeamsJournal journal1 = new TeamsJournal();

researchTeamString1.ResearchTeamChanged += journal1.ResearchTeamChanged;

researchTeamString2.ResearchTeamChanged += journal1.ResearchTeamChanged;

researchTeamString1.AddDefaults();

researchTeamString2.CollectionName = "new collection name";

var rt = new ResearchTeam();

researchTeamString2.AddResearchTeams(rt);

rt.PropertyChanged += journal1.ResearchTeamPropertyHandler;

researchTeamString2.Remove(rt);

rt.PropertyChanged -= journal1.ResearchTeamPropertyHandler;

researchTeamString2.AddResearchTeams(rt);

researchTeamString1.Replace(rt, new ResearchTeam());

researchTeamString1.Remove(new ResearchTeam());

rt.name = "new name";

Console.WriteLine(journal1.ToString());

}

Результат:

