МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

Н.И. Березина

Лабораторные работы по курсу

Объектно-ориентированное программирование: язык программирования C#

Учебное пособие

МАКС Пресс

Москва-2010

УДК 004.43(075.8)
ББК 32.973-018я73
Б48

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики Московского государственного университета имени М.В.Ломоносова

Рецензенты:
Корухова Ю.С. - к.ф.-м.н.
ассистент кафедры алгоритмических языков;

Лопушенко В.В. - к.ф.-м.н.
старший научный сотрудник лаборатории математической физики

Березина Н.И.
Б48 Лабораторные работы по курсу «Объектно-ориентированное программирование: язык программирования C#» : Учебное пособие – М.: Издательский отдел факультета ВМиК МГУ имени М.В.Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2010. – 76 с.
ISBN 978-5-89407-424-5
ISBN 978-5-317-03375-0

В учебном пособии представлены лабораторные работы по практикуму, который сопровождает курс «Объектно-ориентированное программирование: Язык программирования C#». Курс читается в 5 семестре как обязательный кафедральный курс для студентов 3 курса кафедр математической физики, общей математики, функционального анализа и квантовой информатики. Курс также входит в учебный план отделения бакалавров, которые обучаются по направлению “Информационные технологии”.

УДК 004.43(075.8)
ББК 32.973-018я73

ISBN 978-5-89407-424-5
ISBN 978-5-317-03375-0

Введение

В учебном пособии представлены лабораторные работы по практикуму, который сопровождает курс «Объектно-ориентированное программирование: Язык программирования C#». Курс читается в 5 семестре как обязательный кафедральный курс для студентов 3 курса кафедр математической физики, общей математики, функционального анализа и квантовой информатики. Курс также входит в учебный план для бакалавров, которые обучаются по направлению “Информационные технологии”.

Курс является первым из цикла курсов, посвященных платформе Microsoft .NET, его предметом является язык C# как основной язык программирования технологии Microsoft .NET Framework.

Представленная в пособии программа ориентирована на версию .NET Framework 4.0 (3.5), которая поддерживается VisualStudio 2010 (2008).

Программа курса составлена с учетом того, что студенты 3-го курса уже знакомы с объектно-ориентированным языком C++, имеющим много общих синтаксических конструкций с языком C#. По этой причине в курсе акцентируется внимание на различиях в решении проблем, которые возникают при написании управляемого и неуправляемого кода, и больше внимания уделяется конструкциям C#, которых нет в языке C++ - делегатам, событиям, интерфейсам, атрибутам.

Значительная часть курса посвящена изучению типов, определенных в базовых библиотеках Microsoft .NET Framework. Подробно изучается механизм сериализации и определенные в стандартных библиотеках интерфейсы и классы-коллекции, так как они используются в типах, поддерживающих программную модель пользовательского интерфейса, которая изучается в 6 семестре.

Лабораторные работы составлены так, чтобы в процессе работы они давали практические навыки работы со всеми основными конструкциями языка C#.

Для первой лабораторной работы представлены три варианта одного уровня сложности. Лабораторная работа 2 содержит дополнительное задание повышенного уровня сложности. Лабораторные работы 3-5 предлагаются в вариантах двух уровней сложности.

Все пять лабораторных работ каждого из вариантов логически связаны между собой. В первой лабораторной работе определяются типы, связанные между собой отношением агрегации. Во второй лабораторной работе эти типы определяются на основе отношения наследования. В третьей лабораторной работе эти же типы определяются как универсальные. При этом в каждой следующей лабораторной работе к ним добавляются новые функциональные возможности.

Автор пользуется возможностью выразить искреннюю благодарность зам. декана факультета ВМК доц. Березину Б.И., зав. кафедрой математической

физики проф. Денисову А.М. и зав. лабораторией математической физики проф. Дмитриеву В.И. за постоянную поддержку и помощь.

Рекомендуемая литература

1. Троелсен Э. Язык программирования C# 2008 и платформа .NET 3.5- Изд. Вильямс, 2010.
2. Трей Нэш. C# 2010. Ускоренный курс для профессионалов. - Изд. Вильямс, 2010.
3. Нейгел К., Ивьен Б. и др. C# 2008 и платформа NET 3.5 для профессионалов. – Изд. Диалектика, 2008.
4. Герберт Шилдт. C# 3.0. Полное руководство. - Изд. Вильямс, 2010.

Программа курса «Объектно-ориентированное программирование: Язык программирования C#»

Общеязыковая среда выполнения (CLR). Единая система типов .NET Framework

Введение. Общеязыковая среда выполнения (CLR). Базовая библиотека классов (BCL). Сборка (assembly) как минимальная единица повторного использования. Метаданные и манифест сборки. Единая система типов .NET Framework как основа межъязыкового взаимодействия. Типы-значения (value types) и ссылочные типы (reference types). Класс System.Object как общий базовый класс всех типов C#. Упаковка(boxing) и распаковка(unboxing) типов-значений. Анонимные типы. Арифметические типы. Вычисления с плавающей запятой. Допустимые преобразования типов. Тип dynamic.

Массивы и строки

Массивы. Определение и инициализация. Одномерные массивы. Прямоугольные и ступенчатые (jagged) многомерные массивы. Допустимые приведения типов массивов. Классы System.String и System.Text.StringBuilder для работы со строками.

Пространства имен. Классы и структуры

Структура программы на языке C#. Пространства имен. Классы и структуры в C#. Члены класса. Доступ к членам класса (полям и методам). Константы и поля, доступные только для чтения. Статические методы и данные. Инициализация объектов. Конструкторы. Статический конструктор. Статические

классы. Вложенные классы. Частичные классы. Частичные методы. Свойства и индексаторы (свойства с параметрами). Автореализуемые свойства. Инициализаторы объектов. Передача параметров размерных и ссылочных типов. Модификаторы ref и out. Методы с переменным числом параметров. Модификатор params. Перегрузка операторов.

Исключения. Типы с явным освобождением ресурсов. Сборщик мусора

Механизм исключений. Блоки catch и finally. Иерархия библиотечных классов-исключений. Жизненный цикл объекта. Деструкторы и метод Finalize. Сборщик мусора. Типы с явным освобождением ресурсов. Сравнение подходов к решению задачи освобождения ресурсов в языках C# и C++.

Наследование. Полиморфные типы

Наследование. Отношение агрегации и отношение наследования. Наследование и конструкторы. Полиморфные типы. Переопределение виртуальных и невиртуальных методов. Виртуальные свойства. Абстрактные классы. Преобразование типов. Операторы as и is.

Тип интерфейс (interface)

Тип интерфейс - определение и реализация. Явная и неявная реализация интерфейса. Интерфейс как параметр или как тип возвращаемого значения. Реализация интерфейсов и наследование.

Типы-коллекции

Классы-коллекции из пространства имен System.Collections. Итераторы. Блок итератора. Оператор yield. Интерфейсы IEnumerable и IEnumerator. Оператор foreach. Интерфейсы ICollection и IList. Класс ArrayList. Интерфейс IDictionary и класс Hashtable. Интерфейсы IComparable и IComparer.

Универсальные (обобщенные) типы (generics)

Универсальные типы и CLR. Универсальные методы. Ограничения (constraints) для типовых параметров. Универсальные типы и наследование. Обобщения и явное и неявное приведение типов. Статические универсальные методы в System.Array. Тип System.Tuple. Универсальные интерфейсы ICollection<T>, IList<T> и IDictionary<T>. Универсальные классы-коллекции в базовой библиотеке классов. Классы List<T> и Dictionary<TKey,TValue>.

Тип делегат (delegate)

Тип delegate. Цепочки делегатов. Классы System.Delegate и System.MulticastDelegate. Анонимные методы. Обобщенные делегаты.

Тип событие (event)

Определение и реализация событий. Свойства события (event properties). События и интерфейсы. События, определенные в базовой библиотеке классов. Делегаты EventHandler и EventHandler<TEventArgs>. Тип System.EventArgs. Интерфейс System.ComponentModel.**INotifyPropertyChanged**.

Расширяющие методы (extension methods) и язык интегрированных запросов LINQ

Делегаты System.Func<·>. Лямбда-выражения как способ объявления анонимных методов. Выражения запросов. Стандартные операции запросов.

Потоковые классы. Механизм сериализации. Версия сборки

Классы для работы с файлами и каталогами, потоковый класс System.IO.FileStream, классы читатели/писатели System.IO.BinaryReader и System.IO.BinaryWriter. Механизм сериализации. Общая схема сериализации. Нестандартная (пользовательская) сериализация. Версия сборки. Сборки со строгим именем. Толерантность по отношению к внешним или неожидаемым данным, толерантность по отношению к пропущенным данным при сериализации.

Отражение (reflection). Атрибуты

Механизм отражения (reflection) как способ получения из метаданных полной информации о типе в период выполнения. Класс System.Type. Атрибуты. Позиционные и именованные параметры атрибута. Определение пользовательских атрибутов. Атрибуты, определенные в базовой библиотеке классов.

Взаимодействие управляемого и неуправляемого кода

Взаимодействие управляемого и неуправляемого кода. Сервис PInvoke. Атрибуты DllImport и MarshalAs. Маршалинг типов-значений и ссылочных типов. Копирование (copying) и прикрепление (pinning) данных при маршалинге.

Глобализация и локализация приложения

Глобализация и локализация приложения. Региональные настройки (culture). Класс System.Globalization.CultureInfo.

Лабораторная работа 1

Классы, свойства, индексаторы. Одномерные, прямоугольные и ступенчатые массивы

Требования к программе, общие для всех вариантов

Определить класс **Person**, который имеет

- закрытое поле типа string, в котором хранится имя;
- закрытое поле типа string, в котором хранится фамилия;
- закрытое поле типа System.DateTime для даты рождения.

В классе **Person** определить конструкторы:

- конструктор с тремя параметрами типа string, string, DateTime для инициализации всех полей класса;
- конструктор без параметров, инициализирующий все поля класса некоторыми значениями по умолчанию.

В классе **Person** определить свойства с методами get и set:

- свойство типа string для доступа к полю с именем;
- свойство типа string для доступа к полю с фамилией;
- свойство типа DateTime для доступа к полю с датой рождения;
- свойство типа int с методами get и set для получения информации(get) и изменения (set) года рождения в закрытом поле типа DateTime, в котором хранится дата рождения.

В классе **Person** определить

- перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса;
- виртуальный метод string ToShortString(), который возвращает строку, содержащую только имя и фамилию.

Сравнить время, необходимое для выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов.

Для этого в методе Main() создать

- одномерный массив;
- двумерный прямоугольный массив;
- двумерный ступенчатый массив.

Тип элементов массивов зависит от варианта лабораторной работы. Массивы должны иметь одинаковое число элементов. Если число строк в двумерном прямоугольном массиве равно nrow, а число столбцов ncolumn, то одномерный массив должен содержать nrow*ncolumn элементов, в двумерном ступенчатом массиве общее число элементов также должно быть равно nrow*ncolumn.

Значения nrow и ncolumn вводятся в процессе работы приложения в виде одной строки с разделителями. В приглашении, которое получает пользователь, должна быть информация о том, какие символы можно использовать как разделители, число разделителей должно быть больше 1. С помощью метода Split класса System.String приложение разбирает введенную пользователем текстовую строку с информацией о числе строк и числе столбцов двумерного массива и присваивает значения переменным, которые содержат значения nrow и ncolumn. В первой лабораторной работе не требуется обрабатывать ошибки ввода, предполагается, что пользователь правильно ввел данные.

Приложение распределяет память для всех массивов и инициализирует элементы массивов. Для инициализации элементов можно использовать конструктор без параметров.

Для всех элементов массивов выполняется одна и та же операция, например, присваивается значение одному из свойств, определенных для элементов массива. В лабораторной работе требуется сравнить время выполнения этой операции для одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов.

Для измерения времени выполнения операций можно использовать свойство Environment.TickCount. Статическое свойство TickCount класса Environment имеет тип int, использует информацию системного таймера и содержит время в миллисекундах, которое прошло с момента перезагрузки компьютера.

Чтобы получить время выполнения некоторого блока кода, необходимо вызвать Environment.TickCount непосредственно перед блоком и сразу же после последнего оператора блока и взять разность значений.

В блоке кода, для которого измеряется время, не должно быть операций распределения памяти для массивов, инициализации элементов массивов и операций вывода данных на консоль. Блоки кода должны содержать только операции с элементами массива.

Вычисленные значения времени выполнения операций для трех типов массивов, а также число строк nrow и столбцов ncolumn выводятся на консоль. Вывод должен быть подписан, т.е. вывод должен содержать информацию о том, какому типу массива отвечает выведенное значение.

При работе в среде VisualStudio необходимо уметь

- средствами Solution Explorer добавить в проект новый класс;
- с помощью диаграммы классов(Class Diagram) добавить в класс методы, поля и свойства;
- с помощью диаграммы классов добавить в класс перегруженную (override) версию виртуального метода.

Вариант 1. Требования к программе

Определить тип **Education** - перечисление(enum) со значениями Specialist, Bachelor, SecondEducation.

Определить класс **Exam**, который имеет три открытых автореализуемых свойства, доступных для чтения и записи:

- свойство типа string, в котором хранится название предмета;
- свойство типа int, в котором хранится оценка;
- свойство типа System.DateTime для даты экзамена.

В классе **Exam** определить:

- конструктор с параметрами типа string, int и DateTime для инициализации всех свойств класса;
- конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
- перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех свойств класса.

Определить класс **Student**, который имеет

- закрытое поле типа Person, в котором хранятся данные студента;
- закрытое поле типа Education для информации о форме обучения;
- закрытое поле типа int для номера группы;
- закрытое поле типа Exam [] для информации об экзаменах, которые сдал студент.

В классе **Student** определить конструкторы:

- конструктор с параметрами типа Person, Education, int для инициализации соответствующих полей класса;
- конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

В классе **Student** определить свойства с методами get и set:

- свойство типа Person для доступа к полю с данными студента;
- свойство типа Education для доступа к полю с формой обучения;
- свойство типа int для доступа к полю с номером группы;
- свойство типа Exam [] для доступа к полю со списком экзаменов.

В классе **Student** определить

- свойство типа double (только с методом get), в котором вычисляется средний балл как среднее значение оценок в списке сданных экзаменов;
- индексатор булевского типа (только с методом get) с одним параметром типа Education; значение индексатора равно true, если значение поля с

формой обучения студента совпадает со значением индекса, и false в противном случае;

- метод void AddExams (params Exam []) для добавления элементов в список экзаменов;
- перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список экзаменов;
- виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка экзаменов, но со значением среднего балла.

В методе **Main()**

5. Создать один объект типа Student, преобразовать данные в текстовый вид с помощью метода ToShortString() и вывести данные.
6. Вывести значения индексатора для значений индекса Education.Specialist, Education.Bachelor и Education.SecondEducation.
7. Присвоить значения всем определенным в типе Student свойствам, преобразовать данные в текстовый вид с помощью метода ToString() и вывести данные.
8. С помощью метода AddExams(params Exam[]) добавить элементы в список экзаменов и вывести данные объекта Student, используя метод ToString().
9. Сравнить время выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов типа Exam.

Вариант 2. Требования к программе

Определить тип **Frequency -** перечисление(enum) со значениями Weekly, Monthly, Yearly.

Определить класс **Article**, который имеет три открытых автореализуемых свойства, доступных для чтения и записи:

- свойство типа Person, в котором хранятся данные автора статьи;
- свойство типа string для названия статьи;
- свойство типа double для рейтинга статьи.

В классе **Article** определить:

- конструктор с параметрами типа Person, string, double для инициализации всех свойств класса;

- конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
- перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех свойств класса.

Определить класс **Magazine**, который имеет
- закрытое поле типа string с названием журнала;
- закрытое поле типа Frequency с информацией о периодичности выхода журнала;
- закрытое поле типа DateTime с датой выхода журнала;
- закрытое поле типа int с тиражом журнала;
- закрытое поле типа Article[] со списком статей в журнале.

В классе **Magazine** определить конструкторы:
- конструктор с параметрами типа string, Frequency, DateTime, int для инициализации соответствующих полей класса;
- конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

В классе **Magazine** определить свойства с методами get и set:
- свойство типа string для доступа к полю с названием журнала;
- свойство типа Frequency для доступа к полю с информацией о периодичности выхода журнала;
- свойство типа DateTime для доступа к полю с датой выхода журнала;
- свойство типа int для доступа к полю с тиражом журнала;
- свойство типа Article[] для доступа к полю со списком статей.

В классе **Magazine** определить
- свойство типа double (только с методом get), в котором вычисляется среднее значение рейтинга в списке статей;
- индексатор булевского типа (только с методом get) с одним параметром типа Frequency; значение индексатора равно true, если значение поля типа Frequency совпадает со значением индекса, и false в противном случае;
- метод void AddArticles (params Article[]) для добавления элементов в список статей в журнале;
- перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список статей;
- виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка статей, но со значением среднего рейтинга статей.

В методе **Main()**

1. Создать один объект типа Magazine, преобразовать данные в текстовый вид с помощью метода ToShortString() и вывести данные.
2. Вывести значения индексатора для значений индекса Frequency.Weekly, Frequency.Monthly и Frequency.Yearly.
3. Присвоить значения всем определенным в типе Magazine свойствам, преобразовать данные в текстовый вид с помощью метода ToString() и вывести данные.
4. С помощью метода AddArticles(params Article[]) добавить элементы в список статей и вывести данные объекта Magazine, используя метод ToString().
5. Сравнить время выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов типа Article.

Вариант 3. Требования к программе

Определить тип **TimeFrame** - перечисление(enum) со значениями Year, TwoYears, Long.

Определить класс **Paper**, который имеет три открытых автореализуемых свойства, доступных для чтения и записи:

- свойство типа string, в котором хранится название публикации;
- свойство типа Person для автора публикации;
- свойство типа DateTime с датой публикации.

В классе **Paper** определить

- конструктор с параметрами типа string, Person, DateTime для инициализации всех свойств класса;
- конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
- перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса.

Определить класс **ResearchTeam**, который имеет

- закрытое поле типа string с названием темы исследований;
- закрытое поле типа string с названием организации;
- закрытое поле типа int – регистрационный номер;
- закрытое поле типа TimeFrame для информации о продолжительности исследований;
- закрытое поле типа Paper[], в котором хранится список публикаций.

В классе **ResearchTeam** определить конструкторы:

- конструктор с параметрами типа string, string, int, TimeFrame для инициализации соответствующих полей класса;
- конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

В классе **ResearchTeam** определить свойства с методами get и set:

- свойство типа string для доступа к полю с названием темы исследований;
- свойство типа string для доступа к полю с названием организации;
- свойство типа int для доступа к полю с номером регистрации;
- свойство типа TimeFrame для доступа к полю с продолжительностью исследований;
- свойство типа Paper[] для доступа к полю со списком публикаций по теме исследований.

В классе **ResearchTeam** определить

- свойство типа Paper (только с методом get), которое возвращает ссылку на публикацию с самой поздней датой выхода; если список публикаций пустой, свойство возвращает значение null;
- индексатор булевского типа (только с методом get) с одним параметром типа TimeFrame; значение индексатора равно true, если значение поля с информацией о продолжительности исследований совпадает со значением индекса, и false в противном случае;
- метод void AddPapers (params Paper[]) для добавления элементов в список публикаций;
- перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список публикаций;
- виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка публикаций.

В методе **Main()**

1. Создать один объект типа ResearchTeam, преобразовать данные в текстовый вид с помощью метода ToShortString() и вывести данные.
2. Вывести значения индексатора для значений индекса TimeFrame.Year, TimeFrame.TwoYears, TimeFrame.Long.
3. Присвоить значения всем определенным в типе ResearchTeam свойствам, преобразовать данные в текстовый вид с помощью метода ToString() и вывести данные.
4. С помощью метода AddPapers (params Paper []) добавить элементы в список публикаций и вывести данные объекта ResearchTeam.

5. Вывести значение свойства, которое возвращает ссылку на публикацию с самой поздней датой выхода;
6. Сравнить время выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов типа Paper.

Лабораторная работа 2

Наследование. Исключения. Интерфейсы. Итераторы и блоки итераторов

Информация для всех вариантов

В классе **Person** из лабораторной работы 1 и в классах, дополнительно указанных в вариантах, надо

- переопределить (override) виртуальный метод bool Equals (object obj);
- определить операции == и != ;
- переопределить виртуальный метод int GetHashCode();

Реализация виртуального метода bool Equals (object obj) в классе System.Object определяет равенство объектов как равенство ссылок на объекты. Некоторые классы из базовой библиотеки BCL переопределяют метод Equals(). В классе System.String этот метод переопределен так, что равными считаются строки, которые совпадают посимвольно. Реализация метода Equals() в структурном типе DateTime равенство объектов DateTime определяет как равенство значений.

В лабораторной работе требуется переопределить метод Equals так, чтобы объекты считались равными, если равны все данные объектов. Для класса Person это означает, что равны даты рождения и посимвольно совпадают строки с именем и фамилией.

Определение операций == и != должно быть согласовано с переопределенным методом Equals, т.е. критерии, по которым проверяется равенство объектов в методе Equals, должны использоваться и при проверке равенства объектов в операциях == и !=.

Переопределение виртуального метода int GetHashCode() также должно быть согласовано с операциями == и !=. Виртуальный метод GetHashCode() используется некоторыми классами базовой библиотеки, например, коллекциями-словарями. Классы базовой библиотеки, вызывающие метод GetHashCode() из пользовательского типа, предполагают, что равным объектам отвечают равные значения хэш-кодов. Поэтому в случае, когда под равенством объектов понимается совпадение данных (а не ссылок), реализация метода

GetHashCode() должна для объектов с совпадающими данными возвращать равные значения хэш-кодов.

В классах, указанных в вариантах лабораторной работы, требуется определить метод object DeepCopy() для создания полной копии объекта. Определенные в некоторых классах базовой библиотеки методы Clone() и Copy() создают ограниченную (shallow) копию объекта – при копировании объекта копии создаются только для полей структурных типов, для полей ссылочных типов копируются только ссылки. В результате в ограниченной копии объекта поля-ссылки указывают на те же объекты, что и в исходном объекте.

Метод DeepCopy() должен создать полные копии всех объектов, ссылки на которые содержат поля типа. После создания полная копия не зависит от исходного объекта - изменение любого поля или свойства исходного объекта не должно приводить к изменению копии.

При реализации метода DeepCopy() в классе, который имеет поле типа System.Collections.ArrayList, следует иметь в виду, что определенные в классе ArrayList конструктор ArrayList(ICollection) и метод Clone() при создании копии коллекции, состоящей из элементов ссылочных типов, копируют только ссылки.

Метод DeepCopy() должен создать как копии элементов коллекции ArrayList, так и полные копии объектов, на которые ссылаются элементы коллекции. Для типов, содержащих коллекции, реализация метода DeepCopy() упрощается, если в типах элементов коллекций также определить метод DeepCopy().

Вариант 1. Требования к программе

Определить интерфейс

```
interface IDateAndCopy
    { object DeepCopy();
      DateTime Date { get; set; }
    }
```

Определить новые версии классов **Exam**, **Person** и **Student** из лабораторной работы 1. В классы **Exam**, **Person** и **Student** добавить реализацию интерфейса IDateAndCopy. Новую версию класса **Student** определить как класс, производный от класса **Person**.

Все поля новой версии класса **Person** определить с доступом protected, сохранить все свойства, определенные в первой версии класса.

В новой версии класса **Person** дополнительно

- переопределить метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Person трактовалось как совпадение всех данных объектов, а не ссылок на объекты Person;
- переопределить виртуальный метод int GetHashCode();
- определить виртуальный метод object DeepCopy();
- реализовать интерфейс IDateAndCopy.

Определить класс **Test**, который имеет два открытых автореализуемых свойства, доступных для чтения и записи:

- свойство типа string, в котором хранится название предмета;
- свойство типа bool для информации о том, сдан зачет или нет.

В классе **Test** определить:

- конструктор с параметрами типа string и bool для инициализации свойств класса;
- конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
- перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех свойств класса.

Класс **Student** определить как производный от класса **Person**.

Новая версия класса **Student** имеет следующие поля:

- закрытое поле типа Education для информации о форме обучения;
- закрытое поле типа int для номера группы;
- закрытое поле типа System.Collections.ArrayList, в котором хранится список зачетов (объекты типа **Test**);
- закрытое поле типа System.Collections.ArrayList для списка экзаменов (объекты типа **Exam**).

Код следующих конструкторов, методов и свойств из старой версии класса **Student** необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс Person, и в новой версии класса Student список экзаменов хранится в коллекции System.Collections.ArrayList:

- конструктор с параметрами типа Person, Education, int для инициализации соответствующих полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойство типа Person; метод get свойства возвращает объект типа Person, данные которого совпадают с данными подобъекта базового класса, метод set присваивает значения полям из подобъекта базового класса;
- свойство типа double (только с методом get), в котором вычисляется средний балл как среднее значение оценок в списке сданных экзаменов;

- свойство типа System.Collections.ArrayList с методами get и set для доступа к полю со списком экзаменов;
- метод void AddExams (params Exam[]) для добавления элементов в список экзаменов;
- перегруженная версия виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список зачетов и экзаменов;
- виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка зачетов и экзаменов, но со значением среднего балла.

Дополнительно в новой версии класса **Student**

- определить перегруженную версию виртуального метода object DeepCopy();
- реализовать интерфейс IDateAndCopy;
- определить свойство типа int с методами get и set для доступа к полю с номером группы. В методе set бросить исключение, если присваиваемое значение меньше или равно 100 или больше 599. При создании объекта-исключения использовать один из определенных в библиотеке CLR классов-исключений, инициализировать объект-исключение с помощью конструктора с параметром типа string, в сообщении передать информацию о допустимых границах для значения свойства.

В новой версии класса **Student** определить

- итератор для последовательного перебора всех элементов (объектов типа object) из списков зачетов и экзаменов (объединение);
- итератор с параметром для перебора экзаменов (объектов типа Exam) с оценкой больше заданного значения.

В методе **Main()**

1. Создать два объекта типа Person с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэш-кодов для объектов.
2. Создать объект типа Student, добавить элементы в список экзаменов и зачетов, вывести данные объекта Student.
3. Вывести значение свойства типа Person для объекта типа Student.
4. С помощью метода DeepCopy() создать полную копию объекта Student. Изменить данные в исходном объекте Student и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.

5. В блоке try/catch присвоить свойству с номером группы некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.
6. С помощью оператора foreach для итератора, определенного в классе Student, вывести список всех зачетов и экзаменов.
7. С помощью оператора foreach для итератора с параметром, определенного в классе Student, вывести список всех экзаменов с оценкой выше 3.

Дополнительное задание:

В классе **Student**

- реализовать интерфейс System.Collections.IEnumerable для перебора названий всех предметов (объектов типа string), которые есть как в списке зачетов, так и в списке экзаменов (пересечение). Для этого определить вспомогательный класс StudentEnumerator, реализующий интерфейс System.Collections.IEnumerator.
- 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;
- 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.

В методе **Main()**

8. С помощью оператора foreach для объекта типа Student вывести список предметов, которые есть как в списке зачетов, так и в списке экзаменов.
9. С помощью оператора foreach для итератора, определенного в классе Student, вывести список всех сданных зачетов и сданных экзаменов.
10. С помощью оператора foreach для итератора, определенного в классе Student, вывести список сданных зачетов, для которых сдан и экзамен.

Вариант 2. Требования к программе

Определить интерфейс

```
interface IRateAndCopy
    { double Rating { get;}
      object DeepCopy();
```

}

Определить новые версии классов **Person**, **Article** и **Magazine** из лабораторной работы 1. Класс **Magazine** определить как производный от класса **Edition**. В классы **Article** и **Magazine** добавить реализацию интерфейса IRateAndCopy.

В новой версии класса **Person** дополнительно

- переопределить метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Person трактовалось как совпадение всех данных объектов, а не ссылок на объекты Person;
- переопределить виртуальный метод int GetHashCode();
- определить виртуальный метод object DeepCopy().

В новой версии класса **Article** дополнительно

- определить виртуальный метод object DeepCopy();
- реализовать интерфейс IRateAndCopy.

Определить класс **Edition**. Класс **Edition** имеет

- защищенное(protected) поле типа string с названием издания;
- защищенное поле типа DateTime с датой выхода издания;
- защищенное поле типа int с тиражом издания;

В классе **Edition** определить:

- конструктор с параметрами типа string, DateTime, int для инициализации соответствующих полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойства с методами get и set для доступа к полям типа;
- виртуальный метод object DeepCopy();
- свойство типа int с методами get и set для доступа к полю с тиражом издания; в методе set свойства бросить исключение, если присваиваемое значение отрицательно. При создании объекта-исключения использовать один из определенных в библиотеке CLR классов-исключений, инициализировать объект-исключение с помощью конструктора с параметром типа string, в сообщении передать информацию о допустимых значениях свойства.

В классе **Edition** переопределить (override):

- виртуальный метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Edition трактовалось как совпадение всех данных объектов, а не ссылок на объекты Edition;
- виртуальный метод int GetHashCode();
- перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса.

Новая версия класса **Magazine** имеет базовый класс **Edition** и следующие поля:

- закрытое поле типа Frequency с информацией о периодичности выхода журнала;
- закрытое поле типа System.Collections.ArrayList со списком редакторов журнала (объектов типа **Person**).
- закрытое поле типа System.Collections.ArrayList, в котором хранится список статей в журнале (объектов типа **Article**).

Код следующих конструкторов, методов и свойств из старой версии класса **Magazine** необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс Edition, и в новой версии класса Magazine для списка статей используется тип System.Collections.ArrayList:

- конструктор с параметрами типа string, Frequency, DateTime, int для инициализации соответствующих полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойство типа double (только с методом get), в котором вычисляется среднее значение рейтинга статей в журнале;
- свойство типа System.Collections.ArrayList для доступа к полю со списком статей в журнале;
- метод void AddArticles (params Article[]) для добавления элементов в список статей в журнале;
- перегруженная версия виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список статей и список редакторов;
- виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка статей и списка редакторов, но со значением среднего рейтинга статей в журнале.

Дополнительно в новой версии класса **Magazine** реализовать

- свойство типа System.Collections.ArrayList для доступа к списку редакторов журнала;
- метод void AddEditors (params Person[]) для добавления элементов в список редакторов;
- перегруженную (override) версию виртуального метода object DeepCopy();
- интерфейс IRateAndCopy;
- свойство типа Edition; метод get свойства возвращает объект типа Edition, данные которого совпадают с данными подобъекта базового класса, метод set присваивает значения полям из подобъекта базового класса.

В новой версии класса **Magazine** определить

- итератор с параметром типа double для перебора статей с рейтингом больше некоторого заданного значения;
- итератор с параметром типа string для перебора статей, в названии которых есть заданная строка.

В методе **Main()**

1. Создать два объекта типа Edition с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэш-кодов для объектов.
2. В блоке try/catch присвоить свойству с тиражом издания некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.
3. Создать объект типа Magazine, добавить элементы в списки статей и редакторов журнала и вывести данные объекта Magazine.
4. Вывести значение свойства типа Edition для объекта типа Magazine.
5. С помощью метода DeepCopy() создать полную копию объекта Magazine. Изменить данные в исходном объекте Magazine и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.
6. С помощью оператора foreach для итератора с параметром типа double вывести список всех статей с рейтингом больше некоторого заданного значения.
7. С помощью оператора foreach для итератора с параметром типа string вывести список статей, в названии которых есть заданная строка.

Дополнительное задание:

В классе **Magazine**

- реализовать интерфейс System.Collections.IEnumerable для перебора статей (объектов типа Article), авторы которых не входят в список редакторов журнала; для этого определить вспомогательный класс MagazineEnumerator, реализующий интерфейс System.Collections.IEnumerator.
- определить итератор для перебора статей (объектов типа Article), авторы которых являются редакторами журнала, для этого определить метод, содержащий блок итератора и использующий оператор yield.
- определить итератор для перебора редакторов журнала (объектов типа Person), у которых нет статей в журнале, для этого определить метод, содержащий блок итератора и использующий оператор yield.

В методе **Main()**

8. С помощью оператора foreach для объекта типа Magazine вывести список статей, авторы которых не являются редакторами журнала.
9. С помощью оператора foreach для итератора, определенного в классе Magazine, вывести список статей, авторы которых являются редакторами журнала.
10. С помощью оператора foreach для итератора, определенного в классе Magazine, вывести список редакторов, у которых нет статей в журнале.

Вариант 3. Требования к программе

Определить интерфейс

```
interface INameAndCopy
    { string Name { get; set;}
      object DeepCopy();
    }
```

Определить новые версии классов **Person**, **Paper** и **ResearchTeam** из лабораторной работы 1. Класс **ResearchTeam** определить как производный от класса **Team**. В классы **Team** и **ResearchTeam** добавить реализацию интерфейса INameAndCopy.

В классе **Paper** определить виртуальный метод object DeepCopy().

В новой версии класса **Person** дополнительно

- переопределить метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Person трактовалось как совпадение всех данных объектов, а не ссылок на объекты Person;
- переопределить виртуальный метод int GetHashCode();
- определить виртуальный метод object DeepCopy().

Определить класс **Team**. Класс **Team** имеет

- защищенное (protected) поле типа string с названием организации;
- защищенное поле типа int – регистрационный номер.

В классе **Team** определить:

- конструктор с параметрами типа string и int для инициализации полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойство типа string для доступа к полю с названием организации;
- свойство типа int для доступа к полю с номером регистрации; в методе set бросить исключение, если присваиваемое значение меньше или равно 0;

при создании объекта-исключения использовать один из определенных в библиотеке CLR классов-исключений, инициализировать объект-исключение с помощью конструктора с параметром типа string.

В классе **Team**

- определить виртуальный метод object DeepCopy();
- реализовать интерфейс INameAndCopy.

В классе **Team** переопределить (override):

- виртуальный метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Team трактовалось как совпадение всех данных объектов, а не ссылок на объекты Team;
- виртуальный метод int GetHashCode();
- виртуальный метод string ToString() для формирования строки со значениями всех полей класса.

Новая версия класса **ResearchTeam** имеет базовый класс **Team** и следующие поля:

- закрытое поле типа string с названием темы исследований;
- закрытое поле типа TimeFrame с информацией о продолжительности исследований;
- закрытое поле типа System.Collections.ArrayList со списком участников проекта (объектов типа Person);
- закрытое поле типа System.Collections.ArrayList для списка публикаций (объектов типа Paper).

Код следующих конструкторов, методов и свойств из старой версии класса **ResearchTeam** необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс **Team**, и в новой версии класса ResearchTeam для списка публикаций используется тип System.Collections.ArrayList:

- конструктор с параметрами типа string, string, int, TimeFrame для инициализации соответствующих полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойство типа System.Collections.ArrayList для доступа к полю со списком публикаций;
- свойство типа Paper (только с методом get), которое возвращает ссылку на публикацию с самой поздней датой выхода; если список публикаций пустой, свойство возвращает значение null;
- метод void AddPapers (params Paper[]) для добавления элементов в список публикаций;
- перегруженная версия виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список публикаций и список участников проекта;

- метод string ToShortString(), который формирует строку со значениями всех полей класса без списка публикаций и списка участников проекта.

Дополнительно в новой версии класса **ResearchTeam** определить

- перегруженную версию виртуального метода object DeepCopy();
- свойство типа System.Collections.ArrayList для доступа к полю со списком участников проекта;
- метод void AddMembers (params Person[]) для добавления элементов в список участников проекта;
- 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;
- реализовать интерфейс INameAndCopy.

В новой версии класса **ResearchTeam** определить

- итератор для последовательного перебора участников проекта (объектов типа Person), не имеющих публикаций;
- итератор с параметром типа int для перебора публикаций, вышедших за последние n лет, в котором число n передается через параметр итератора.

В методе **Main()**

1. Создать два объекта типа Team с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэш-кодов для объектов.
2. В блоке try/catch присвоить свойству с номером регистрации некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.
3. Создать объект типа ResearchTeam, добавить элементы в список публикаций и список участников проекта и вывести данные объекта ResearchTeam.
4. Вывести значение свойства Team для объекта типа ResearchTeam.
5. С помощью метода DeepCopy() создать полную копию объекта ResearchTeam. Изменить данные в исходном объекте ResearchTeam и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.
6. С помощью оператора foreach для итератора, определенного в классе ResearchTeam, вывести список участников проекта, которые не имеют публикаций.

7. С помощью оператора foreach для итератора с параметром, определенного в классе ResearchTeam, вывести список всех публикаций, вышедших за последние два года.

Дополнительное задание:

В классе ResearchTeam

- реализовать интерфейс System.Collections.IEnumerable для перебора участников проекта (объектов типа Person), у которых есть публикации; для этого определить вспомогательный класс ResearchTeamEnumerator, реализующий интерфейс System.Collections.IEnumerator.
- определить итератор для перебора участников проекта (объектов типа Person), имеющих более одной публикации, для этого определить метод, содержащий блок итератора и использующий оператор yield.
- определить итератор для перебора публикаций (объектов типа Paper), вышедших за последний год, для этого определить метод, содержащий блок итератора и использующий оператор yield.

В методе **Main()**

8. С помощью оператора foreach для объекта типа ResearchTeam вывести список участников проекта, у которых есть публикации.
9. С помощью оператора foreach для итератора, определенного в классе ResearchTeam, вывести список участников проекта, имеющих более одной публикации.
10. С помощью оператора foreach для итератора, определенного в классе ResearchTeam, вывести список публикаций, вышедших за последний год.

Лабораторная работа 3. Варианты первого уровня

Универсальные типы. Классы-коллекции. Методы расширения класса System.Linq.Enumerable

Информация для всех вариантов

Во всех вариантах лабораторной работы 3 требуется определить класс **TestCollections**, который содержит поля следующих типов

- System.Collections.Generic.List<TKey> ;
- System.Collections.Generic.List<string> ;
- System.Collections.Generic.Dictionary<TKey, TValue> ;

- System.Collections.Generic.Dictionary<string, TValue> .

Конкретные значения типовых параметров TKey и TValue зависят от варианта. Во всех вариантах тип ключа TKey и тип значения TValue связаны отношением базовый-производный. Во всех вариантах в классе TValue определено свойство, которое возвращает ссылку на объект типа TKey с данными, совпадающими с данными подобъекта базового класса (это свойство должно возвращать ссылку на объект типа TKey, а не ссылку на вызывающий объект TValue).

В конструкторе класса **TestCollections** создаются коллекции с заданным числом элементов. Надо сравнить время поиска элемента в коллекциях-списках List<TKey> и время поиска элемента по ключу и элемента по значению в коллекциях-словарях Dictionary<TKey,TValue>.

Для автоматической генерации элементов коллекций в классе TestCollections надо определить статический метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект типа TValue.

Каждый объект TValue содержит подобъект базового класса TKey. Соответствие между значениями целочисленного параметра метода и подобъектами TKey класса TValue должно быть взаимно-однозначным – равным значениям параметра должны отвечать равные объекты TKey и наоборот. Равенство объектов типа TKey трактуется так же, как это было сделано в лабораторной работе 2 при определении операций равенства объектов.

Все четыре коллекции содержат одинаковое число элементов. Каждому элементу из коллекции List<TKey> должен отвечать элемент в коллекции Dictionary<TKey, TValue> с равным значением ключа. Список List<string> состоит из строк, которые получены в результате вызова метода ToString() для объектов TKey из списка List<TKey>. Каждому элементу списка List<string> отвечает элемент в коллекции-словаре Dictionary<string, TValue> с равным значением ключа типа string.

Число элементов в коллекциях вводится пользователем в процессе работы приложения. Если при вводе была допущена ошибка, приложение должно обработать исключение, сообщить об ошибке ввода и повторить прием ввода до тех пор, пока не будет правильно введено целочисленное значение.

Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска

- элемента в коллекциях List<TKey> и List<string> с помощью метода Contains;
- элемента по ключу в коллекциях Dictionary< TKey, TValue> и Dictionary <string, TValue > с помощью метода ContainsKey;
- значения элемента в коллекции Dictionary< TKey, TValue > с помощью метода ContainsValue.

Так как статический метод для автоматической генерации элементов должен обеспечивать взаимно-однозначное соответствие между значением целочисленного параметра метода и объектами TKey, этот метод можно использовать как при создании коллекций с большим числом элементов, так и для генерации элемента для поиска.

Вариант 1. Требования к программе

Определить новые версии классов **Person** и **Student** из лабораторной работы 2.

В класс **Person** добавить реализацию интерфейсов

- System.IComparable для сравнения объектов типа Person по полю с фамилией;
- System.Collections.Generic.IComparer<Person> для сравнения объектов типа Person по дате рождения.

В новой версии класса **Student** для списков зачетов и экзаменов использовать типы

- System.Collections.Generic.List<Test> для списка зачетов;
- System.Collections.Generic.List<Exam> для списка экзаменов.

В новой версии класса **Student** сохранить все остальные поля, свойства и методы из предыдущей версии класса, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списков зачетов и экзаменов.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Student>, который можно использовать для сравнения объектов типа Student по среднему баллу.

Определить класс **StudentCollection**, который содержит

- закрытое поле типа System.Collections.Generic.List<Student>;
- метод void AddDefaults(), с помощью которого можно добавить некоторое число элементов типа Student для инициализации коллекции по умолчанию;
- метод void AddStudents (params Student[]) для добавления элементов в список List<Student>;
- перегруженную версию виртуального метода string ToString() для формирования строки с информацией обо всех элементах списка List<Student>, включающую значения всех полей, список зачетов и экзаменов для каждого элемента Student;

- метод string ToShortString(), который формирует строку с информацией обо всех элементах списка List<Student>, содержащую значения всех полей, средний балл, число зачетов и число экзаменов для каждого элемента Student, но без списков зачетов и экзаменов.

В классе **StudentCollection** определить методы, выполняющие сортировку списка List<Student>

- по фамилии студента с использованием интерфейса IComparable, реализованного в классе Person;
- по дате рождения студента с использованием интерфейса IComparer<Person>, реализованного в классе Person;
- по среднему баллу с использованием интерфейса IComparer<Student>, реализованного во вспомогательном классе.

В классе **StudentCollection** определить свойства и методы, выполняющие операции со списком List<Student> с использованием методов расширения класса System.Linq.Enumerable, и статические методы-селекторы, которые необходимы для выполнения соответствующих операций со списком:

- свойство типа double (только с методом get), возвращающее максимальное значение среднего балла для элементов списка List<Student>; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего балла использовать метод Max класса System.Linq.Enumerable;
- свойство типа IEnumerable<Student> (только с методом get), возвращающее подмножество элементов списка List<Student> с формой обучения Education.Specialist; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;
- метод List<Student> AverageMarkGroup(double value), который возвращает список, в который входят элементы Student из списка List<Student> с заданным значением среднего балла; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

Определить класс **TestCollections**, в котором в качестве типа TKey используется класс Person, а в качестве типа TValue - класс Student. Класс содержит закрытые поля с коллекциями типов

- System.Collections.Generic.List<Person>;
- System.Collections.Generic.List<string>;
- System.Collections.Generic.Dictionary <Person, Student>;
- System.Collections.Generic.Dictionary <string, Student>.

В классе **TestCollections** определить

- статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа Student и используется для автоматической генерации элементов коллекций;
- конструктор с параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;
- метод, который вычисляет время поиска элемента в списках List<Person> и List<string>, время поиска элемента по ключу и время поиска элемента по значению в коллекциях-словарях Dictionary<Person, Student> и Dictionary<string, Student>.

В методе **Main()**

1. Создать объект типа StudentCollection. Добавить в коллекцию несколько различных элементов типа Student и вывести объект StudentCollection.
2. Для созданного объекта StudentCollection вызвать методы, выполняющие сортировку списка List<Student> по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку
   - по фамилии студента;
   - по дате рождения;
   - по среднему баллу.
3. Вызвать методы класса StudentCollection, выполняющие операции со списком List<Student>, и после каждой операции вывести результат операции. Выполнить
   - вычисление максимального значения среднего балла для элементов списка;
   - фильтрацию списка для отбора студентов с формой обучения Education.Specialist;
   - группировку элементов списка по значению среднего балла; вывести все группы элементов.
4. Создать объект типа TestCollections. Вызвать метод для поиска в коллекциях первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев. Вывод должен содержать информацию о том, к какой коллекции и к какому элементу относится данное значение.

Вариант 2. Требования к программе

Определить новые версии классов **Edition** и **Magazine** из лабораторной работы 2.

В новой версии класса **Magazine** использовать типы

- System.Collections.Generic.List<Person> для списка редакторов журнала;
- System.Collections.Generic.List<Article> для списка статей в журнале.

В новых версиях классов **Edition** и **Magazine** сохранить все остальные поля, свойства и методы из предыдущей версии класса, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списка редакторов и списка статей.

В класс **Edition** добавить реализацию

- интерфейса System.IComparable для сравнения объектов Edition по полю с названием издания;
- интерфейса System.Collections.Generic.IComparer<Edition> для сравнения объектов Edition по дате выхода издания.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Edition>, который можно использовать для сравнения объектов типа Edition по тиражу издания.

Определить класс **MagazineCollection**, который содержит

- закрытое поле типа System.Collections.Generic.List<Magazine>;
- метод void AddDefaults (), с помощью которого в список List<Magazine> можно добавить некоторое число элементов типа Magazine для инициализации коллекции по умолчанию;
- метод void AddMagazines (params Magazine []) для добавления элементов в список List<Magazine>;
- перегруженную версию виртуального метода string ToString() для формирования строки с информацией обо всех элементах списка List<Magazine>, в том числе значения всех полей, список редакторов журнала и список статей в журнале для каждого элемента Magazine;
- виртуальный метод string ToShortString(), который формирует строку с информацией обо всех элементах списка List<Magazine>, содержащую значения всех полей, средний рейтинг статей, число редакторов журнала и число статей в журнале для каждого элемента Magazine, но без списков редакторов и статей.

В классе **MagazineCollection** определить свойства и методы, выполняющие сортировку списка List<Magazine>

- по названию издания с использованием интерфейса IComparable, реализованного в классе Edition;
- по дате выхода издания с использованием интерфейса IComparer<Edition>, реализованного в классе Edition;

- по тиражу издания с использованием интерфейса IComparer<Edition>, реализованного во вспомогательном классе.

В классе **MagazineCollection** определить методы, выполняющие операции со списком List<Magazine> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций с коллекциями:

- свойство типа double (только с методом get), возвращающее максимальное значение среднего рейтинга статей для элементов списка List<Magazine>; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего рейтинга статей надо использовать метод Max класса System.Linq.Enumerable;
- свойство типа IEnumerable<Magazine> (только с методом get), возвращающее подмножество элементов списка List<Magazine> с периодичностью выхода журнала Frequency.Monthly; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;
- метод List<Magazine> RatingGroup(double value), который возвращает список, содержащий элементы Magazine из List<Magazine> со средним рейтингом статей, который больше или равен value; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

Определить класс **TestCollections**, в котором в качестве типа TKey используется класс Edition, а в качестве типа TValue - класс Magazine. Класс содержит закрытые поля с коллекциями типов

- System.Collections.Generic.List<Edition>;
- System.Collections.Generic.List<string>;
- System.Collections.Generic.Dictionary <Edition, Magazine>;
- System.Collections.Generic.Dictionary <string, Magazine>.

В классе **TestCollection** определить

- статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа Magazine и используется для автоматической генерации элементов коллекций;
- конструктор с параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;
- метод, который вычисляет время поиска элемента в списках List<Edition> и List<string>, время поиска элемента по ключу и время поиска элемента по значению в коллекциях-словарях Dictionary< Edition, Magazine> и Dictionary<string, Magazine>.

В методе **Main()**

1. Создать объект типа MagazineCollection. Добавить в коллекцию несколько элементов типа Magazine с разными значениями полей и вывести объект MagazineCollection.
2. Для созданного объекта MagazineCollection вызвать методы, выполняющие сортировку списка List<Magazine> по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку
   - по названию издания;
   - по дате выхода издания;
   - по тиражу издания.
3. Вызвать методы класса MagazineCollection, выполняющие операции со списком List<Magazine>, и после каждой операции вывести результат операции. Выполнить
   - вычисление максимального значения среднего рейтинга статей для элементов списка; вывести максимальное значение;
   - фильтрацию списка для отбора журналов с периодичностью выхода Frequency.Monthly, вывести результат фильтрации;
   - группировку элементов списка по значению среднего рейтинга статей; вывести все группы элементов.
4. Создать объект типа TestCollections. Вызвать метод для поиска в коллекциях первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев. Вывод должен содержать информацию о том, к какой коллекции и к какому элементу относится данное значение.

Вариант 3. Требования к программе

Определить новые версии классов **Team** и **ResearchTeam** из лабораторной работы 2.

В новой версии класса **ResearchTeam** использовать типы

- System.Collections.Generic.List<Person>для списка участников проекта;
- System.Collections.Generic.List<Paper> для списка публикаций;

В новых версиях классов **Team** и **ResearchTeam** сохранить все остальные поля, свойства и методы из предыдущих версий, внести необходимые исправления в код свойств и методов из-за изменения типа полей для списков.

В новую версию класса **Team** добавить реализацию интерфейса System.IComparable для сравнения объектов Team по полю с номером регистрации.

В новую версию класса **ResearchTeam** добавить реализацию интерфейса System.Collections.Generic.IComparer<ResearchTeam> для сравнения объектов ResearchTeam по названию темы исследований.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<ResearchTeam>, который можно использовать для сравнения объектов типа ResearchTeam по числу публикаций.

Определить класс **ResearchTeamCollection**, который содержит

- закрытое поле типа System.Collections.Generic.List<ResearchTeam>;
- метод void AddDefaults (), с помощью которого в список List<ResearchTeam> можно добавить некоторое число элементов типа ResearchTeam для инициализации коллекции по умолчанию;
- метод void AddResearchTeams (params ResearchTeam []) для добавления элементов в список List<ResearchTeam>;
- перегруженную версию виртуального метода string ToString() для формирования строки с информацией обо всех элементах списка List<ResearchTeam>, которая содержит значения всех полей, список участников проекта и список публикаций для каждого элемента ResearchTeam;
- виртуальный метод string ToShortString(), который формирует строку с информацией обо всех элементах списка List<ResearchTeam>, включающую значения всех полей, число участников проекта и число публикаций для каждого элемента ResearchTeam, но без списков участников и публикаций.

В классе **ResearchTeamCollection** определить методы, выполняющие сортировку списка List<ResearchTeam>

- по номеру регистрации с использованием интерфейса IComparable, реализованного в классе Team;
- по названию темы исследований с использованием интерфейса IComparer<ResearchTeam>, реализованного в классе ResearchTeam;
- по числу публикаций с использованием интерфейса IComparer<ResearchTeam>, реализованного во вспомогательном классе.

В классе **ResearchTeamCollection** определить свойства и методы, выполняющие операции со списком List<ResearchTeam> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций со списком:

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

если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска минимального значения номера регистрации надо использовать метод Min класса System.Linq.Enumerable;

- 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;
- 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.

Определить класс **TestCollections**, в котором в качестве типа TKey используется класс Team, а в качестве типа TValue - класс ResearchTeam. Класс содержит закрытые поля с коллекциями типов

- System.Collections.Generic.List<Team>;
- System.Collections.Generic.List<string>;
- System.Collections.Generic.Dictionary <Team, ResearchTeam>;
- System.Collections.Generic.Dictionary <string, ResearchTeam>.

В классе **TestCollections** определить

- статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа ResearchTeam и используется для автоматической генерации элементов коллекций;
- конструктор с параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;
- метод, который вычисляет время поиска элемента в списках List<Team> и List<string>, время поиска элемента по ключу и время поиска значения элемента в коллекциях-словарях Dictionary<Team, ResearchTeam> и Dictionary <string, ResearchTeam>.

В методе **Main()**

1. Создать объект типа ResearchTeamCollection. Добавить в коллекцию несколько элементов типа ResearchTeam с разными значениями полей и вывести объект ResearchTeamCollection.
2. Для созданного объекта ResearchTeamCollection вызвать методы, выполняющие сортировку списка List<ResearchTeam> по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку
    - по номеру регистрации;

- по названию темы исследований;
- по числу публикаций.

3. Вызвать методы класса ResearchTeamCollection, выполняющие операции со списком List<ResearchTeam>, и после каждой операции вывести результат операции. Выполнить
   - вычисление минимального значения номера регистрации для элементов списка; вывести минимальное значение;
   - фильтрацию проектов с продолжительностью исследований TimeFrame.TwoYears, вывести результат фильтрации;
   - группировку элементов списка по числу публикаций; вывести все группы элементов из списка.
4. Создать объект типа TestCollections. Вызвать метод для поиска в коллекциях первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев. Вывод должен содержать информацию о том, к какой коллекции и к какому элементу относится данное значение.

Лабораторная работа 3. Варианты второго уровня

Универсальные типы. Классы-коллекции. Методы расширения класса System.Linq.Enumerable

Информация для всех вариантов

Во всех вариантах второго уровня требуется определить универсальный делегат

```
delegate System.Collections.Generic.KeyValuePair<TKey,TValue>
         GenerateElement<TKey, TValue>(int j);
```

и универсальный класс **TestCollections<TKey,TValue>**, который содержит закрытые поля следующих типов

- System.Collections.Generic.List<TKey>;
- System.Collections.Generic.List<string> ;
- System.Collections.Generic.Dictionary<TKey, TValue> ;
- System.Collections.Generic.Dictionary<string, TValue>;
- GenerateElement<TKey, TValue>.

Конкретные значения типовых параметров TKey и TValue зависят от варианта.

В конструкторе класса **TestCollections<TKey,TValue>** создаются коллекции с заданным числом элементов. Надо сравнить время поиска элемента в коллекциях-списках List<TKey> и время поиска элемента по ключу и элемента по значению в коллекциях-словарях Dictionary<TKey,TValue>.

Для автоматической генерации элементов коллекций надо определить метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект типа KeyValuePair<TKey,TValue>. Метод должен инициализировать объекты KeyValuePair<TKey,TValue> так, чтобы соответствие между номером элемента и объектом TKey в паре ключ-значение было взаимно-однозначным.

Метод для автоматической генерации элементов коллекций передается в класс TestCollections<TKey,TValue> через параметр конструктора класса. Для этого в классе TestCollections<TKey,TValue> надо определить конструктор с двумя параметрами, имеющими тип int и GenerateElement. Через целочисленный параметр объектам класса передается число элементов в коллекциях, через экземпляр делегата GenerateElement – метод, который используется для автоматической генерации пары ключ-значение в виде объекта KeyValuePair<TKey,TValue>.

Число элементов в коллекциях пользователь вводит в процессе работы приложения. Если при вводе была допущена ошибка, приложение должно обработать исключение, сообщить об ошибке ввода и повторить прием ввода до тех пор, пока не будет правильно введено целочисленное значение.

Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию, – надо измерить время поиска

- элемента в коллекциях List<TKey> и List<string> с помощью метода Contains;
- элемента по ключу в коллекциях Dictionary< TKey, TValue> и Dictionary <string, TValue > с помощью метода ContainsKey;
- значения элемента в коллекции Dictionary< TKey, TValue > с помощью метода ContainsValue.

Так как статический метод для автоматической генерации элементов должен обеспечивать взаимно-однозначное соответствие между значением целочисленного параметра метода и объектами TKey, его можно использовать как при создании коллекций, так и для генерации элемента для поиска.

Вариант 1. Требования к программе

Определить новые версии классов **Exam** и **Student** из лабораторной работы 2.

В класс **Exam** добавить реализацию интерфейсов

- System.IComparable для сравнения объектов типа Exam по названию предмета;

- System.Collections.Generic.IComparer<Exam> для сравнения объектов типа Exam по оценке.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Exam>, который можно использовать для сравнения объектов типа Exam по дате экзамена.

В новой версии класса **Student** для списков зачетов и экзаменов использовать типы

- System.Collections.Generic.List<Test> для списка зачетов;
- System.Collections.Generic.List<Exam> для списка экзаменов.

В новой версии класса **Student** сохранить все остальные поля, свойства и методы из предыдущей версии класса, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списков.

В классе **Student** определить методы для сортировки списка экзаменов

- по названию предмета;
- по оценке;
- по дате экзамена.

Определить универсальный делегат

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

Определить универсальный класс **StudentCollection<TKey>**, содержащий коллекцию объектов Student, в котором для хранения коллекции используется тип System.Collections.Generic.Dictionary<TKey, Student>. Типовой параметр TKey универсального класса StudentCollection<TKey> определяет тип ключа в коллекции Dictionary<TKey, Student>.

Метод, который используется для вычисления ключа при добавлении элемента Student в коллекцию класса StudentCollection<TKey>, отвечает делегату KeySelector<TKey> и передается StudentCollection<TKey> через параметр единственного конструктора класса.

Класс StudentCollection<TKey> содержит

- закрытое поле типа System.Collections.Generic.Dictionary<TKey, Student>;
- закрытое поле типа KeySelector<TKey> для хранения экземпляра делегата с методом, вычисляющим ключ для объекта Student;
- конструктор с одним параметром типа KeySelector<TKey> ;
- метод void AddDefaults (), с помощью которого можно добавить некоторое число элементов типа Student для инициализации коллекции по умолчанию;

- метод void AddStudents (params Student[]) для добавления элементов в коллекцию Dictionary<TKey, Student>;
- перегруженную версию виртуального метода string ToString() для формирования строки, содержащей информацию обо всех элементах коллекции Dictionary<TKey, Student>, в том числе значения всех полей класса Student, включая список зачетов и экзаменов;
- метод string ToShortString(), который 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.

В классе **StudentCollection<TKey>** определить свойства и методы, выполняющие операции со словарем Dictionary<TKey,Student> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций с коллекцией:

- свойство типа double (только с методом get), возвращающее максимальное значение среднего балла для элементов Dictionary<TKey,Student>; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего балла надо использовать метод Max класса System.Linq.Enumerable;
- метод IEnumerable<KeyValuePair<TKey,Student>>EducationForm(Education value), возвращающий подмножество элементов коллекции Dictionary<TKey,Student> с заданной формой обучения; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;
- свойство типа IEnumerable<IGrouping<Education,KeyValuePair<TKey,Student>>> (только с методом get), выполняющее группировку элементов коллекции Dictionary<TKey, Student> в зависимости от формы обучения студента с помощью метода Group класса System.Linq.Enumerable.

В методе **Main()**

1. Создать объект Student и вызвать методы, выполняющие сортировку списка экзаменов List<Exam> по разным критериям, после каждой сортировки вывести данные объекта. Выполнить сортировку
    - по названию предмета;
    - по оценке;
    - по дате экзамена.

2. Создать объект типа StudentCollection<string>. Добавить в коллекцию несколько разных элементов типа Student и вывести объект StudentCollection<string>.
3. Вызвать методы класса StudentCollection<string>, выполняющие операции с коллекцией-словарем Dictionary<TKey, Student>, и после каждой операции вывести результат операции. Выполнить
    - вычисление максимального значения среднего балла для элементов коллекции; вывести максимальное значение;
    - вызвать метод EducationForm для выбора студентов с заданной формой обучения, вывести результат фильтрации;
    - вызвать свойство класса, выполняющее группировку элементов коллекции по форме обучения; вывести все группы элементов.
4. Создать объект типа TestCollection<Person, Student>. Ввести число элементов в коллекциях и вызвать метод для поиска первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев.

Вариант 2. Требования к программе

Определить новые версии классов **Article, Edition** и **Magazine** из лабораторной работы 2.

В класс **Article** добавить реализации интерфейсов

- System.IComparable для сравнения объектов типа Article по названию статьи;
- System.Collections.Generic.IComparer<Article> для сравнения объектов типа Article по фамилии автора.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Article>, который можно использовать для сравнения объектов типа Article по рейтингу статьи.

В новой версии класса **Magazine** использовать типы

- System.Collections.Generic.List<Person> для списка редакторов журнала;
- System.Collections.Generic.List<Article> для списка статей в журнале.

В новых версиях **Edition** и **Magazine** сохранить все остальные поля, свойства и методы из предыдущей версии класса, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списка редакторов журнала и списка статей.

В классе **Magazine** определить методы для сортировки списка статей

- по названию статьи;
- по фамилии автора;
- по рейтингу статьи.

Определить универсальный делегат

```
delegate TKey KeySelector<TKey>(Magazine mg);
```

Определить универсальный класс **MagazineCollection<TKey>**, содержащий коллекцию объектов типа Magazine, в котором для хранения коллекции используется тип System.Collections.Generic.Dictionary<TKey, Magazine>. Типовой параметр TKey универсального класса MagazineCollection<TKey> определяет тип ключа в коллекции Dictionary<TKey, Magazine>.

Метод, который используется для вычисления ключа при добавлении элемента Magazine в коллекцию класса MagazineCollection<TKey>, отвечает делегату KeySelector<TKey> и передается MagazineCollection<TKey> через параметр единственного конструктора класса.

Класс **MagazineCollection<TKey>** содержит

- закрытое поле типа System.Collections.Generic.Dictionary<TKey, Magazine>;
- закрытое поле типа KeySelector<TKey> для хранения экземпляра делегата с методом, вычисляющим ключ для объекта Magazine;
- конструктор с одним параметром типа KeySelector<TKey> ;
- метод void AddDefaults(), с помощью которого можно добавить некоторое число элементов типа Magazine для инициализации коллекции по умолчанию;
- метод void AddMagazines (params Magazine[]) для добавления элементов в коллекцию Dictionary<TKey, Magazine>;
- перегруженную версию виртуального метода string ToString() для формирования строки, содержащей информацию обо всех элементах коллекции Dictionary<TKey, Magazine>, в том числе значения всех полей, включая список редакторов издания и список статей в журнале для каждого элемента Magazine;
- метод string ToShortString(), который формирует строку с информацией обо всех элементах коллекции Dictionary<TKey, Magazine>, содержащую значения всех полей, значение среднего рейтинга статей, число редакторов издания и число статей в журнале для каждого элемента Magazine, но без списков редакторов и статей.

В классе **MagazineCollection<TKey>** определить свойства и методы, выполняющие операции со словарем Dictionary<TKey, Magazine> с использованием методов расширения класса System.Linq.Enumerable и

статические методы-селекторы, которые необходимы для выполнения соответствующих операций с коллекцией:

- свойство типа double (только с методом get), возвращающее максимальное значение среднего рейтинга статей для элементов коллекции; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего рейтинга статей надо использовать метод Max класса System.Linq.Enumerable;
- метод IEnumerable<KeyValuePair<TKey,Magazine>>FrequencyGroup(Frequency value), возвращающий подмножество элементов коллекции Dictionary<TKey,Magazine> с заданной периодичностью выхода журнала; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;
- свойство типа IEnumerable<IGrouping<Frequency,KeyValuePair<TKey,Magazine >>> (только с методом get), выполняющее группировку элементов коллекции Dictionary<TKey, Magazine> в зависимости от периодичности выхода журнала с помощью метода Group класса System.Linq.Enumerable.

В методе **Main()**

1. Создать объект Magazine и вызвать методы, выполняющие сортировку списка List<Article> статей в журнале по разным критериям, после каждой сортировки вывести данные объекта. Выполнить сортировку
   - по названию статьи;
   - по фамилии автора;
   - по рейтингу статьи.
2. Создать объект MagazineCollection<string>. Добавить в коллекцию несколько разных элементов типа Magazine и вывести объект MagazineCollection<string>.
3. Вызвать методы класса MagazineCollection<string>, выполняющие операции с коллекцией-словарем Dictionary<TKey, Magazine>, и после каждой операции вывести результат операции. Выполнить
   - вычисление максимального значения среднего рейтинга статей для элементов коллекции;
   - вызвать метод FrequencyGroup для выбора журналов с заданной периодичностью выхода;
   - вызвать свойство класса, выполняющее группировку элементов коллекции по периодичности выхода; вывести все группы элементов.

4. Создать объект типа TestCollection<Edition, Magazine>. Ввести число элементов в коллекциях и вызвать метод для поиска первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев.

Вариант 3. Требования к программе

Определить новые версии классов **Paper, Team** и **ResearchTeam** из лабораторной работы 2.

В класс **Paper** добавить реализацию интерфейсов

- System.IComparable для сравнения объектов типа Paper по дате выхода публикации;
- System.Collections.Generic.IComparer<Paper> для сравнения объектов типа Paper по названию публикации.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Paper>, который можно использовать для сравнения объектов типа Paper по фамилии автора публикации.

В новой версии класса **ResearchTeam** использовать типы

- System.Collections.Generic.List<Person> для списка участников проекта;
- System.Collections.Generic.List<Paper> для списка публикаций.

В новых версиях классов **Team** и **ResearchTeam** сохранить все остальные поля, свойства и методы из предыдущих версий, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списка участников проекта и списка публикаций.

В классе **ResearchTeam** определить методы для сортировки списка публикаций

- по дате выхода публикации;
- по названию публикации;
- по фамилии автора.

Определить универсальный делегат

```
delegate TKey KeySelector<TKey>( ResearchTeam rt);
```

Определить универсальный класс **ResearchTeamCollection<TKey>**, содержащий коллекцию объектов типа ResearchTeam, в котором для хранения коллекции используется тип System.Collections.Generic.Dictionary<TKey, ResearchTeam>.

Типовой параметр TKey универсального класса ResearchTeamCollection<TKey> определяет тип ключа в коллекции Dictionary<TKey, ResearchTeam>.

Метод, который используется для вычисления ключа при добавлении элемента ResearchTeam в коллекцию класса ResearchTeamCollection<TKey>, отвечает делегату KeySelector<TKey> и передается ResearchTeamCollection<TKey> через параметр единственного конструктора класса.

Класс **ResearchTeamCollection<TKey>** содержит

- закрытое поле типа System.Collections.Generic.Dictionary<TKey, ResearchTeam>;
- закрытое поле типа KeySelector<TKey> для хранения экземпляра делегата с методом, вычисляющим ключ для объекта ResearchTeam;
- конструктор с одним параметром типа KeySelector<TKey> ;
- метод void AddDefaults (), с помощью которого можно добавить некоторое число элементов ResearchTeam для инициализации коллекции по умолчанию;
- метод void AddResearchTeams (params ResearchTeam []) для добавления элементов в коллекцию Dictionary<TKey, ResearchTeam>;
- перегруженную версию виртуального метода string ToString() для формирования строки, содержащей информацию обо всех элементах коллекции Dictionary<TKey, ResearchTeam>, в том числе значения всех полей, включая список участников проекта и список публикаций для каждого элемента ResearchTeam;
- метод string ToShortString(), который формирует строку с информацией обо всех элементах коллекции Dictionary<TKey, ResearchTeam>, содержащую значения всех полей, число участников проекта и число публикаций для каждого элемента ResearchTeam, но без списков участников и публикаций.

В классе **ResearchTeamCollection<TKey>** определить свойства и методы, выполняющие операции со словарем Dictionary<TKey, ResearchTeam> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций с коллекцией:

- свойство типа DateTime (только с методом get), возвращающее дату последней по времени выхода публикации среди всех элементов коллекции; если в коллекции нет элементов, свойство возвращает значение по умолчанию для типа DateTime; для поиска максимального значения среднего рейтинга статей надо использовать метод Max класса System.Linq.Enumerable;
- метод IEnumerable<KeyValuePair<TKey,ResearchTeam>>TimeFrameGroup (TimeFrame value), возвращающий подмножество элементов коллекции Dictionary<TKey, ResearchTeam> со значением продолжительности исследований, которое передается как параметр; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

- свойство типа IEnumerable<IGrouping<TimeFrame,KeyValuePair<TKey,ResearchTeam>>> (только с методом get), выполняющее группировку элементов коллекции Dictionary<TKey, ResearchTeam> в зависимости от продолжительности исследований с помощью метода Group класса System.Linq.Enumerable.

В методе **Main()**

1. Создать объект ResearchTeam и вызвать методы, выполняющие сортировку списка публикаций List<Paper> по разным критериям, после каждой сортировки вывести данные объекта. Выполнить сортировку
   - по дате выхода публикации;
   - по названию публикации;
   - по фамилии автора.
2. Создать объект ResearchTeamCollection<string>. Добавить в коллекцию несколько разных элементов ResearchTeam и вывести объект ResearchTeamCollection<string>.
3. Вызвать методы класса ResearchTeamCollection<string>, выполняющие операции с коллекцией-словарем Dictionary<TKey, ResearchTeam>, после каждой операции вывести результат операции. Выполнить
   - поиск даты последней по времени выхода публикации среди всех элементов коллекции;
   - вызвать метод TimeFrameGroup для выбора объектов ResearchTeam с заданным значением продолжительности исследований;
   - вызвать свойство класса, выполняющее группировку элементов коллекции по значениию продолжительности исследований; вывести все группы элементов из списка.
4. Создать объект типа TestCollection<Team, ResearchTeam>. Ввести число элементов в коллекциях и вызвать метод для поиска первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев.

Лабораторная работа 4. Варианты первого уровня

Делегаты. События

Информация для всех вариантов

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

Коллекция состоит из объектов ссылочных типов. Коллекция изменяется при удалении/добавлении элементов или при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение. В этом случае в соответствующих методах или свойствах класса бросаются события.

При изменении данных объектов, ссылки на которые входят в коллекцию, значения самих ссылок не изменяются. Этот тип изменений не порождает событий.

Для событий, извещающих об изменениях в коллекции, определяется свой делегат. События регистрируются в специальных классах-слушателях.

Вариант 1. Требования к программе

Определить новую версию класса **StudentCollection** из лабораторной работы 3, которая с помощью событий сообщает об изменениях в коллекции.

Для событий определить делегат **StudentListHandler** с сигнатурой:

```
void StudentListHandler
     (object source, StudentListHandlerEventArgs args);
```

Класс **StudentListHandlerEventArgs**, производный от класса System.EventArgs, содержит

- открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
- открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
- открытое автореализуемое свойство типа Student для ссылки на объект, с которым связаны изменения;
- конструкторы для инициализации класса;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.

В новую версию класса **StudentCollection** добавить

- открытое автореализуемое свойство типа string с названием коллекции;
- метод bool Remove (int j) для удаления элемента с номером j из списка List<Student>; если в списке нет элемента с номером j, метод возвращает значение false;
- индексатор типа Student (с методами get и set) с целочисленным индексом для доступа к элементу списка List<Student> с заданным номером.

В новую версию класса **StudentCollection** добавить два события типа StudentListHandler

- **StudentsCountChanged**, которое происходит при добавлении нового элемента в коллекцию или при удалении элемента из коллекции; через объект StudentListHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что в коллекцию был добавлен новый элемент или из нее был удален элемент, ссылку на добавленный или удаленный элемент Student;
- **StudentReferenceChanged**, которое происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; через объект StudentListHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент Student.

Событие StudentsCountChanged бросают следующие методы класса StudentCollection

- AddDefaults();
- AddStudents (params Student[]) ;
- Remove (int j).

Событие StudentReferenceChanged бросает метод set индексатора, определенного в классе StudentCollection.

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

Класс **JournalEntry** содержит

- открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
- открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;

- открытое автореализуемое свойство типа string с данными объекта Student, с которым связаны изменения в коллекции;
- конструктор для инициализации полей класса;
- перегруженную версию метода string ToString().

Класс **Journal** содержит

- закрытое поле типа System.Collections.Generic.List<JournalEntry>;
- обработчики событий StudentsCountChanged и StudentReferenceChanged, которые добавляют новый элемент JournalEntry в список List<JournalEntry>; для инициализации JournalEntry используется информация из объекта StudentListHandlerEventArgs, который передается вместе с событием;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<JournalEntry>.

В методе **Main()**

1. Создать две коллекции StudentCollection.
2. Создать два объекта типа Journal, один объект Journal подписать на события StudentsCountChanged и StudentReferenceChanged из первой коллекции StudentCollection, другой объект Journal подписать на события StudentReferenceChanged из обеих коллекций StudentCollection.
3. Внести изменения в коллекции StudentCollection
   - добавить элементы в коллекции;
   - удалить некоторые элементы из коллекций;
   - присвоить некоторым элементам коллекций новые значения.
4. Вывести данные обоих объектов Journal.

Вариант 2. Требования к программе

Определить новую версию класса **MagazineCollection** из лабораторной работы 3, которая с помощью событий сообщает об изменениях в коллекции.

Для событий определить делегат **MagazineListHandler** с сигнатурой:

```
void MagazineListHandler
     (object source, MagazineListHandlerEventArgs args);
```

Класс **MagazineListHandlerEventArgs**, производный от класса System.EventArgs, содержит

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

- открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
- открытое автореализуемое свойство типа int с номером элемента, который был изменен;
- конструкторы для инициализации класса;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.

В новую версию класса **MagazineCollection** добавить

- открытое автореализуемое свойство типа string с названием коллекции;
- метод bool Replace (int j, Magazine mg) для замены элемента с номером j из списка List<Magazine> на элемент mg; если в списке нет элемента с номером j, метод возвращает значение false;
- индексатор типа Magazine (с методами get и set) с целочисленным индексом для доступа к элементу списка List<Magazine> с заданным номером.

В новую версию класса **MagazineCollection** добавить два события типа MagazineListHandler

- **MagazineAdded**, которое происходит при добавлении элемента в коллекциию; событие передает через объект MagazineListHandlerEventArgs имя коллекции, строку с информацией о том, что в коллекцию был добавлен элемент, и номер добавленного элемента в списке List<Magazine>;
- **MagazineReplaced**, которое происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; событие передает через объект MagazineListHandlerEventArgs имя коллекции, строку с информацией о том, что в коллекции был заменен элемент, и номер замененного элемента.

Событие MagazineAdded бросают методы класса MagazineCollection

- AddDefaults();
- AddMagazines (params Magazine[]) ;

Событие MagazineReplaced бросают

- метод Replace (int j, Magazine mg);
- метод set индексатора, определенного в классе MagazineCollection.

Определить класс **Listener** для накопления информации об изменениях в коллекциях MagazineCollection. В классе Listener информация хранится в списке из элементов типа **ListEntry,** каждый элемент списка содержит информацию об отдельном изменении в коллекции MagazineCollection.

Класс **ListEntry** содержит

- открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
- открытое автореализуемое свойство типа string с информацией о том, какое событие произошло в коллекции;
- номер добавленного или измененного элемента;
- конструктор для инициализации полей класса;
- перегруженную версию метода string ToString().

Класс **Listener** содержит

- список изменений System.Collections.Generics.List<ListEntry>;
- обработчик событий MagazineAdded и MagazineReplaced, который на основе информации из объекта MagazineListHandlerEventArgs, создает элемент ListEntry и добавляет его в список изменений;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<ListEntry>.

В методе **Main()**

1. Создать две коллекции MagazineCollection.
2. Создать два объекта типа Listener, один объект Listener подписать на события MagazineAdded и MagazineReplaced из первой коллекции MagazineCollection, другой объект Listener подписать на события MagazineAdded из обеих коллекций MagazineCollection.
3. Внести изменения в коллекции MagazineCollection
   - добавить элементы в коллекции;
   - заменить некоторые элементы из коллекций с помощью метода Replace класса MagazineCollection;
   - присвоить некоторым элементам коллекций новые значения с помощью индексатора класса MagazineCollection.
4. Вывести данные обоих объектов Listener.

Вариант 3. Требования к программе

Определить новую версию класса **ResearchTeamCollection** из лабораторной работы 3, которая с помощью событий сообщает об изменениях в коллекции.

Для событий определить делегат **TeamListHandler** с сигнатурой:

```
void TeamListHandler
     (object source, TeamListHandlerEventArgs args);
```

Класс **TeamListHandlerEventArgs**, производный от класса System.EventArgs, содержит

- открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
- открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
- открытое автореализуемое свойство типа int с номером элемента, который был добавлен или заменен;
- конструкторы для инициализации класса;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.

В новую версию класса **ResearchTeamCollection** добавить

- открытое автореализуемое свойство типа string с названием коллекции;
- 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;
- индексатор типа ResearchTeam (с методами get и set) с целочисленным индексом для доступа к элементу списка List<ResearchTeam> с заданным номером.

В новую версию класса **ResearchTeamCollection** добавить два события типа TeamListHandler

- **ResearchTeamAdded**, которое происходит при добавлении элемента в конец списка List<ResearchTeam>; событие передает через объект TeamListHandlerEventArgs имя коллекции, строку с информацией о том, что в коллекцию был добавлен элемент, и номер добавленного элемента в списке List<ResearchTeam>;
- **ResearchTeamInserted**, которое происходит, когда новый элемент вставляется перед одним из элементов списка List<ResearchTeam>; событие передает через объект TeamListHandlerEventArgs имя коллекции, строку с информацией о том, что в коллекцию был вставлен элемент, и номер нового элемента.

Событие ResearchTeamAdded бросают методы класса ResearchTeamCollection

- AddDefaults();
- AddResearchTeams (params ResearchTeam []);
- InsertAt (int j, ResearchTeam rt), если элемента с номером j нет в списке.

Событие ResearchTeamInserted бросает метод InsertAt (int j, ResearchTeam rt), если элемент с номером j есть в списке List<ResearchTeam>.

Определить класс **TeamsJournal** для накопления информации об изменениях в коллекциях типа ResearchTeamCollection. В классе TeamsJournal информация хранится в списке из элементов типа **TeamsJournalEntry,** каждый элемент

списка содержит информацию об отдельном изменении в коллекции ResearchTeamCollection.

Класс **TeamsJournalEntry** содержит

- открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
- открытое автореализуемое свойство типа string с информацией о том, какое событие произошло в коллекции;
- номер нового элемента;
- конструктор для инициализации полей класса;
- перегруженную версию метода string ToString().

Класс **TeamsJournal** содержит

- закрытое поле List<TeamsJournalEntry> для списка изменений;
- обработчик событий ResearchTeamAdded и ResearchTeamInserted. Обработчик использует информацию, которая передается ему через объект TeamListHandlerEventArgs, создает элемент TeamsJournalEntry и добавляет его в список 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.
2. Создать два объекта типа TeamsJournal, один объект TeamsJournal подписать на события ResearchTeamAdded и ResearchTeamInserted из первой коллекции ResearchTeamCollection, другой объект TeamsJournal подписать на события ResearchTeamInserted из обеих коллекций ResearchTeamCollection.
3. Внести изменения в коллекции ResearchTeamCollection
    - добавить элементы в коллекции;
    - с помощью метода InsertAt (int j, ResearchTeam rt) перед элементом с номером j, который есть в коллекции, вставить новый элемент;
    - вызвать метод InsertAt (int j, ResearchTeam rt) с номером j, которого нет в коллекции.
4. Вывести данные обоих объектов TeamsJournal.

Лабораторная работа 4. Варианты второго уровня

Делегаты. События

Информация 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
              (Object sender,PropertyChangedEventArgs e);
```

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<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.

Вариант 2. Требования к программе

Определить новые версии классов **Edition, Magazine** и **MagazineCollection<TKey>** из лабораторной работы 3.

Новая версия класса **Edition** реализует интерфейс **System.ComponentModel.INotifyPropertyChanged**. Событие **PropertyChanged** из интерфейса System.ComponentModel.INotifyPropertyChanged происходит при изменении значений свойств класса **Edition**, связанных с тиражом и датой выхода издания. 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) **Update** со значениями **Add**, **Replace** и **Property**.

Для события, которое бросают методы новой версии класса MagazineCollection<TKey>, определить универсальный делегат **MagazinesChangedHandler<TKey>** с сигнатурой:

```
void MagazinesChangedHandler<TKey>
   (object source, MagazinesChangedEventArgs<TKey> args);
```

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

- открытое автореализуемое свойство типа string с названием коллекции;
- открытое автореализуемое свойство типа Update с информацией о том, чем вызвано событие, – добавлением нового элемента в коллекцию, заменой элемента в коллекции или изменением данных элемента;
- открытое автореализуемое свойство типа string с названием свойства класса Magazine, которое является источником изменения данных элемента; для событий, порожденных добавлением или заменой элемента, значение свойства – пустая строка;
- открытое автореализуемое свойство типа TKey с ключом элемента, который был добавлен в коллекцию, заменил один из элементов коллекции или элемента, у которого были изменены данные;
- конструктор с параметрами типа string, Update, string и TKey для инициализации значений всех свойств класса;
- перегруженную версию метода string ToString().

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

- открытое автореализуемое свойство типа string с названием коллекции;
- метод bool Replace(Magazine mold, Magazine mnew) для замены в словаре Dictionary<TKey, Magazine> элемента со значением mold на элемент со значением mnew; если в словаре нет элемента со значением mold, метод возвращает значение false;
- событие **MagazinesChanged** типа MagazinesChangedHandler<TKey>, которое происходит при добавлении нового элемента в коллекцию, замене элемента в коллекции или при изменении данных одного из ее элементов.

Определить класс **Listener**, собирающий информацию об изменениях в классе MagazineCollection<TKey>. Класс Listener содержит список из элементов типа **ListEntry**. Каждый элемент ListEntry содержит информацию об отдельном изменении объекта MagazineCollection<TKey>, в результате которого произошло событие MagazinesChanged.

Класс **ListEntry** содержит автореализуемые свойства
- типа string с названием коллекции;
- типа Update с информацией о типе события;
- типа string с названием свойства класса Magazine, 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 **Listener** содержит
- закрытое поле типа System.Collections.Generics.List<ListEntry>;
- обработчик события MagazinesChanged, который на основе информации из объекта MagazinesChangedEventArgs, создает элемент ListEntry и добавляет его к списку;
- перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<ListEntry>.

В методе **Main()**

1. Создать два объекта MagazineCollection<string> с разными названиями.
2. Создать объект типа Listener и подписать его на события MagazinesChanged из обоих объектов MagazineCollection<string>.
3. Внести изменения в MagazineCollection<string>:
    - добавить элементы в коллекции;
    - изменить значения разных свойств элементов, входящих в коллекцию;

- заменить один из элементов коллекции;
- изменить данные в элементе, который был удален из коллекции при замене элемента.

4. Вывести данные объекта Listener.

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 с номером регистрации объекта ResearchTeam для элемента, который был удален, в том числе и при замене элемента, или элемента, данные которого были изменены;

- конструктор с 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, когда изменяется 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**. Каждый элемент 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 с 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.
4. Вывести данные объекта TeamsJournal.

Лабораторная работа 5. Варианты первого уровня

Классы для работы с файлами. Сериализация

Определить новые версии классов **Student** (вариант **1**), **Magazine** (вариант **2**) и **ResearchTeam** (вариант **3**). В сформулированных ниже требованиях для этих классов использовано общее обозначение **T**.

В новые версии классов добавить **экземплярные** методы

- T DeepCopy() для создания полной копии объекта с использованием сериализации;
- bool Save(string filename) для сохранения данных объекта в файле с помощью сериализации;
- bool Load(string filename) для инициализации объекта данными из файла с помощью десериализации;
- bool AddFromConsole() для добавления в один из списков класса нового элемента, данные для которого вводятся с консоли;

и статические методы

- static bool Save(string filename, T obj) для сохранения объекта в файле с помощью сериализации;
- static bool Load(string filename, T obj) для восстановления объекта из файла с помощью десериализации.

В экземплярном методе **T DeepCopy()** вызывающий объект сериализуется в поток MemoryStream. Метод возвращает восстановленный при десериализации объект, который представляет собой полную копию исходного объекта.

Экземплярный метод **bool Save(string filename)** сериализует все данные вызывающего объекта в файл с именем filename. Если файл с именем filename существует, приложение его перезаписывает. Если такого файла нет, приложение его создает. Метод возвращает значение true, если сериализация завершилась успешно, и значение false в противном случае.

Экземплярный метод **bool Load(string filename)** десериализует данные из файла с именем filename и использует их для инициализации вызывающего объекта. Метод возвращает значение true, если инициализация завершилась успешно. Если полностью выполнить инициализацию объекта не удалось, исходные данные объекта должны остаться без изменения. В этом случае метод возвращает значение false.

Статические методы **bool Save(string filename, T obj)** и **bool Load(string filename, T obj)** получают через параметры имя файла и ссылку на объект, для которого выполняется сериализация или восстановление. Методы возвращают значение true, если сериализация/инициализация завершилась успешно, и значение false в противном случае. Если полностью выполнить инициализацию объекта не удалось, исходные данные объекта должны остаться без изменения.

Во всех реализациях методов сохранения/восстановления данных из файла операции открытия файла, сериализации и десериализации данных должны находиться в блоках try-catch-finally.

В методе **bool AddFromConsole()** для добавления нового элемента в один из списков класса T

- пользователь получает приглашение ввести данные в виде одной строки символов с разделителями; приглашение содержит описание формата строки ввода, в том числе информацию о том, какие символы можно использовать в качестве разделителей;
- выполняется разбор данных; операции преобразования данных, которые могут бросить исключение, должны находиться в блоке try-catch;

- если разбор введенных данных был завершен успешно, в список добавляется новый элемент и метод возвращает значение true; в противном случае пользователь получает сообщение о том, что при вводе были допущены ошибки и возвращаемое значение метода равно false.

В **варианте 1** элементы, данные для которых вводятся с консоли, добавляются в список экзаменов System.Collections.Generic.List<Exam>. Вводятся название предмета, оценка и дата экзамена.

В **варианте 2** элементы добавляются в список статей в журнале System.Collections.Generic.List<Article>. Вводятся название статьи, данные автора статьи для объекта типа Person и рейтинг статьи.

В **варианте 3** элементы добавляются в список публикаций System.Collections.Generic.List<Paper>. Вводятся название публикации, данные автора статьи для объекта типа Person и дата публикации.

В методе **Main()**

1. Создать объект типа T с непустым списком элементов, для которого предусмотрен ввод данных с консоли. Создать полную копию объекта с помощью метода, использующего сериализацию, и вывести исходный объект и его копию.
2. Предложить пользователю ввести имя файла:
   - если файла с введенным именем нет, приложение должно сообщить об этом и создать файл;
   - если файл существует, вызвать метод Load(string filename) для инициализации объекта T данными из файла.
3. Вывести объект T.
4. Для этого же объекта T сначала вызвать метод AddFromConsole(), затем метод Save(string filename). Вывести объект T.
5. Вызвать последовательно
   - статический метод Load(string filename, T obj), передав как параметры ссылку на тот же самый объект T и введенное ранее имя файла;
   - метод AddFromConsole();
   - статический метод Save (string filename, T obj).
6. Вывести объект T.

Приложение должно работать в режиме накопления. Если выбирается один и тот же файл для записи, и пользователь вводит данные без ошибок, при

каждом следующем выполнении приложения к списку добавляются два новых элемента. Приложение должно обрабатывать все исключения, которые могут возникнуть из-за ошибок при вводе данных. Независимо от того, корректно были введены данные или при вводе были допущены ошибки, все файловые потоки должны быть закрыты.

Лабораторная работа 5. Варианты второго уровня

Сериализация. Взаимодействие управляемого и неуправляемого кода

В лабораторной работе требуется определить классы для матриц специального вида на языках C# и C++. Классы содержат метод для решения системы линейных алгебраических уравнений, который учитывает специфику матрицы. Эти классы используются для сравнения времени выполнения управляемого кода C# и неуправляемого кода C++. Код C++ компилируется в DLL-библиотеку. Код C# вызывает методы из DLL-библиотеки C++ с помощью сервиса PInvoke. Для сохранения в файле результатов тестирования времени выполнения кодов C# и C++ используется сериализация.

Варианты отличаются типом матриц специального вида, для которых определяются классы C# и C++.

Вариант	Тип матрицы
1	циркулянтная
2	теплицева
3	теплицева симметричная
4	ганкелева
5	трехдиагональная

Класс Matrix

Матрица специального вида полностью определяется либо одной строкой (циркулянтная и симметричная теплицева), либо одной строкой и одним столбцом (теплицева и ганкелева), либо тремя диагоналями (трехдиагональная матрица).

Память для хранения матрицы распределяется в конструкторе. Порядок матрицы передается как параметр конструктора.

В памяти хранится не вся полная матрица, а только информация, необходимая для того, чтобы задать полную матрицу:

- для циркулянтных и теплицевых симметричных матриц в памяти хранится только одна строка;
- для теплицевых и ганкелевых матриц в памяти хранятся только столбец и строка;
- для трехдиагональных матриц в памяти хранятся только диагонали.

За счет этого экономится память для хранения матриц. Полная матрица содержит n^2 элементов, в то время как для рассматриваемых матриц специального вида порядка n в зависимости от типа матрицы достаточно хранить n, $2n$ или $3n$ элементов.

Для элементов матриц лучше использовать тип double, так как арифметические операции с операндами типа float и double выполняются с одинаковой скоростью, но у типа double шире диапазон значений и выше точность при хранении. Длина типа double гарантирует 15-16 десятичных цифр, длина типа float только 7 десятичных цифр. Тип float по сравнению с double имеет преимущество только в экономии памяти при хранении данных.

В классе C++ должны быть определены

- конструктор Matrix (int n) для инициализации матрицы по умолчанию;
- конструктор для пользовательской инициализации матрицы;
- конструктор копирования с прототипом Matrix (const Matrix &);
- деструктор;
- операция присваивания с прототипом Matrix& operator=(const Matrix &);
- метод для решения системы линейных уравнений.

В коде C++ не должно быть утечки памяти, т.е. **вся память**, выделенная динамически с помощью оператора new, **должна быть освобождена** с помощью оператора delete.

Метод для решения системы линейных уравнений должен принимать как параметр массив элементов типа double, содержащий вектор правой части. В реализации метода на C++ вектор решения также необходимо передать через параметр, в этом случае память для него будет и распределяться, и освобождаться в вызываемом методе. При передаче решения через возвращаемое значение метода память приходится распределять в вызываемом методе, а освобождать в вызывающем. В реализации этого метода на C# вектор решения можно передать как возвращаемое значение, так как память освобождается сборщиком мусора.

В классе C# должны быть определены

- конструктор Matrix (int n) для инициализации матрицы по умолчанию;
- конструктор для пользовательской инициализации матрицы;
- метод для решения системы линейных уравнений;
- перегруженная версия виртуального метода ToString().

Конструктор Matrix(int n) используется для автоматической генерации матрицы порядка n при сравнении скорости выполнения кодов C# и C++. Так как время решения системы линейных уравнений определяется только порядком матрицы и не зависит от того, для какой конкретной матрицы и правой части решается система, в конструкторе для автоматической генерации матрицу надо задать так, чтобы при решении системы уравнений не возникало проблем с плохой обусловленностью матрицы. Для этого достаточно на главной диагонали матрицы задать элементы, которые по модулю значительно больше внедиагональных элементов. Не следует использовать методы случайной генерации элементов матрицы, так как этот процесс значительно медленнее, чем простое присваивание значений, кроме того, чтобы получить хорошо обусловленную матрицу, придется специально корректировать значения.

В приложении 2 приведены формулы для решения системы линейных уравнений с теплицевыми матрицами, которые для системы порядка n требуют n^2 арифметических операций. Для этого алгоритма время решения системы линейных уравнений должно возрастать в 4 раза при увеличении порядка матрицы в 2 раза. Это соотношение выполняется только для матриц достаточно большого порядка, так как для матриц малого порядка накладные расходы на вызов метода, распределение и освобождение памяти в методе решения системы линейных уравнений будут сравнимы со временем выполнения арифметических операций.

В методе для решения системы линейных уравнений не должна распределяться память для полной матрицы, так как обратная матрица определяется либо двумя векторами x и y (теплицева и ганкелева матрицы), либо одним вектором (теплицева симметричная и циркулянтная).

Для трехдиагональных матриц зависимость времени решения от порядка матрицы – линейная, это значит, что для матриц большого порядка при увеличении порядка матрицы в 2 раза, время решения системы линейных уравнений также возрастает в 2 раза.

Вызов неуправляемого кода C++ из кода C#

Код C++ надо скомпилировать в DLL-библиотеку, содержащую две глобальные экспортируемые функции, которые вызываются из кода C#.

Первая глобальная экспортируемая функция C++ используется только для сравнения времени выполнения кода C# и кода C++. Эта функция из кода C# получает через параметры только два целочисленных значения - порядок матрицы и число повторов. Так как для матриц небольшого порядка решение системы выполняется быстро и точности системного таймера не хватает для его измерения, решение одной и той же системы линейных уравнений выполняется несколько раз, число повторов задается в коде C# и передается через параметр глобальной экспортируемой функции.

В первой глобальной экспортируемой функции создается объект типа Matrix и с помощью конструктора Matrix(int n) генерируется матрица заданного порядка. В цикле заданное число раз решается система уравнений для одной и той же матрицы с одной и той же правой часью и измеряется время выполнения этого цикла. Правую часть для системы линейных уравнений нужно инициализировать вне цикла.

Для измерения времени выполнения кода C++ можно использовать функцию clock(). Время выполнения кода C++ возвращается в вызывающий код C# либо через параметр глобальной экспортируемой функции, либо как возвращаемое значение.

Вторая глобальная экспортируемая функция C++ получает из кода C# данные, которые определяют матрицу и правую часть. В коде C++ решается система линейных уравнений, решение возвращается в код C# также через параметр этой глобальной экспортируемой функции.

Самый простой способ обмена данными между кодами C# и C++ – обмен данными в виде массивов с элементами типа double через параметры экспортируемой функции.

Для того, чтобы отделить интерфейсную часть кода C# от вычислительной, в коде на C# надо определить метод, который через параметры получает порядок матрицы и число повторов, автоматически генерирует матрицу и правую часть, выполняет вычисления и возвращает время выполнения цикла на C#.

Оба проекта (для C# и C++) надо разместить в одном решении (solution). В коде C# в атрибуте DllImport надо указать относительный путь к файлу с DLL-библиотекой, скомпилированной из исходного кода C++. В этом случае при копировании решения(solution) в другой каталог не потребуется вносить изменения в атрибут DllImport, а при перекомпиляции C++-проекта файл с DLL-библиотекой не надо будет вручную копировать в текущий каталог проекта C#.

Сериализация

В коде C# определить тип **TimeItem** (класс или структуру) для хранения времени выполнения кода для одной пары значений порядка матрицы и числа повторов и класс **TimesList** для хранения всей информации.

Тип **TimeItem** должен иметь поля, в которых хранятся

- порядок матрицы;
- число повторов;
- время выполнения цикла в коде на C#;
- время выполнения цикла в коде на C++;
- коэффициент, равный отношению времени выполнения кода на C# и кода на C++.

В классе **TimesList** должны быть определены

- закрытое поле типа List<TimeItem> - список объектов типа TimeItem;
- открытый метод Add(TimeItem), добавляющий новый объект TimeItem к списку;
- открытый метод Save(string filename) для сохранения списка List<TimeItem> в файле с использованием сериализации;
- открытый метод Load(string filename) для восстановления списка List<TimeItem> из файла с использованием сериализации.

Когда пользователь завершает работу приложения, список List<TimeItem> из объекта TimesList сохраняется в файле с именем, которое пользователь ввел в начале работы приложения.

В методе **Main()**

1. Проверить, что метод решения системы линейных уравнений работает правильно. Для этого надо программно задать матрицу 3-го порядка и правую часть. В вариантах 1-4 необходимо задать матрицу с несовпадающими элементами в строках и столбцах, которые ее определяют. Диагонали трехдиагональной матрицы также должны содержать несовпадающие элементы. Решить систему линейных уравнений и вывести матрицу, правую часть и решение, полученное в коде C#.
2. Передать в код C++ через параметры глобальной экспортируемой функции данные, которые определяют матрицу и правую часть. Решить систему линейных уравнений, решение передать в код C# и вывести полученное решение.
3. Создать один объект типа TimesList и предложить пользователю ввести имя файла:

- если файла с заданным именем нет, то в объекте TimesList создается пустая коллекция List<TimeItem>;
- если файл уже существует, то приложение
    - открывает файл;
    - инициализирует список List<TimeItem> объекта TimesList данными из файла, выполняя десериализацию;
    - закрывает файл;
    - выводит все данные на экран.

4. После того, как пользователь ввел имя файла, он получает приглашение ввести порядок матрицы и число повторов или завершить работу приложения.
5. Если пользователь ввел порядок матрицы и число повторов,
    - приложение выполняет вычисления на C#;
    - вызывает экспортируемую функцию из DLL-библиотеки C++;
    - сохраняет результаты в объекте TimeItem;
    - добавляет объект TimeItem в список List<TimeItem> объекта TimesList;
    - затем пользователь снова получает приглашение ввести новые значения порядка матрицы и числа повторов или завершить работу приложения.
6. Когда пользователь завершает работу приложения
    - выводится вся коллекция из объекта TimesList;
    - коллекция из объекта TimesList сохраняется с использованием сериализации в файле с именем, которое пользователь ввел в начале работы приложения.
7. Весь вывод должен быть подписан. Элементы списка имеют тип TimeItem, который содержит пять полей. Можно вывести список в виде таблицы, в которой первая строка содержит заголовки с описанием данных, размещенных в столбце, и каждый из элементов списка вывести как строку таблицы.
8. Все исключения, которые могут возникнуть при работе приложения, в том числе из-за ошибок при вводе данных, должны быть обработаны. В частности, код, в котором выполняется сериализация/десериализация и код, вызывающий глобальные экспортируемые функции из DLL-библиотеки C++, должен находиться в блоках try/catch.

Приложение 1

Метод прогонки для решения системы линейных уравнений с трехдиагональной матрицей

Метод прогонки для решения системы линейных уравнений с трехдиагональной матрицей представляет собой метод исключения Гаусса, который учитывает ленточную структуру матрицы системы.

Применяя метод исключения Гаусса к системе линейных уравнений с трехдиагональной матрицей

$$\begin{pmatrix} c_1 & b_1 & 0 & 0 & . & . & 0 & 0 & 0 & 0 \\ a_2 & c_2 & b_2 & 0 & . & . & 0 & 0 & 0 & 0 \\ 0 & a_3 & c_3 & b_3 & . & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & a_{N-2} & c_{N-2} & b_{N-2} & 0 \\ 0 & 0 & 0 & 0 & . & . & 0 & a_{N-1} & c_{N-1} & b_{N-1} \\ 0 & 0 & 0 & 0 & . & . & 0 & 0 & a_N & c_N \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ x_{N-2} \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \\ f_{N-2} \\ f_{N-1} \\ f_N \end{pmatrix}$$

получим следующие рекуррентные формулы для вычисления прогоночных коэффициентов

$$\beta_1 = \frac{b_1}{c_1}, \quad \gamma_1 = \frac{f_1}{c_1}$$

$$\beta_i = \frac{b_i}{c_i - \beta_{i-1} a_i}, \quad \gamma_i = \frac{f_i - a_i \gamma_{i-1}}{c_i - \beta_{i-1} a_i}, \quad 2 \le i \le N-1$$

$$\gamma_N = \frac{f_N - a_N \gamma_{N-1}}{c_N - \beta_{N-1} a_N}$$

и рекуррентные формулы для нахождения решения

$$x_N = \gamma_N$$

$$x_i = \gamma_i - \beta_i x_{i+1}, \quad i = (N-1), (N-2), \ldots 1.$$

Приложение 2

Теплицевы, циркулянтные и ганкелевы матрицы

Матрицы специального вида, например, теплицевы, циркулянтные и ганкелевы, встречаются при численном решении некоторых задач математической физики.

Методы решения систем линейных уравнений, которые учитывают специфику таких матриц, дают возможность значительно уменьшить число арифметических операций по сравнению с методами для матриц общего вида. Так при решении системы линейных алгебраических уравнений с полной матрицей общего вида порядка n необходимо выполнить $O\left(n^3\right)$ арифметических операций, в то время как асимптотически оптимальные методы решения систем уравнений с теплицевой или ганкелевой матрицей дают возможность найти решение за $O(n\log_2^2 n)$ операций, а оптимальные методы решения системы уравнений с циркулянтной матрицей требуют лишь $O(n\log_2 n)$ арифметических операций.

Матрица, элементы которой a_{ij} зависят только от разности индексов $i-j$, называется ***теплицевой:***

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdot & a_{-n+2} & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \cdot & a_{-n+3} & a_{-n+2} \\ a_2 & a_1 & a_0 & \cdot & a_{-n+4} & a_{-n+3} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdot & a_0 & a_{-1} \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdot & a_1 & a_0 \end{bmatrix}$$

Здесь и в дальнейшем использованы обозначения, принятые в [1].

На каждой параллельной главной диагонали теплицевой матрицы расположены равные элементы. Таким образом, теплицева матрица полностью определяется первой строкой и первым столбцом.

Симметричная теплицева матрица определяется одним своим столбцом и имеет следующий вид:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdot & a_{n-2} & a_{n-1} \\ a_1 & a_0 & a_1 & \cdot & a_{n-3} & a_{n-2} \\ a_2 & a_1 & a_0 & \cdot & a_{n-4} & a_{n-3} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdot & a_0 & a_1 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdot & a_1 & a_0 \end{bmatrix}$$

Сумма теплицевых матриц является теплицевой.

Произведение двух теплицевых матриц в общем случае не является теплицевой матрицей. Теплицевость сохраняется лишь в некоторых частных случаях, например, в случае циркулянтных матриц или верхних или нижних треугольных теплицевых матриц.

Теплицева матрица называется ***циркулянтной*** (или циркулянтом), если для всех $i \neq 0$ выполняется соотношение $a_{-i} = a_{n-i}$. Циркулянтная матрица полностью определяется одной своей строкой и имеет следующий вид:

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdot & a_{-n+2} & a_{-n+1} \\ a_{-n+1} & a_0 & a_{-1} & \cdot & a_{-n+3} & a_{-n+2} \\ a_{-n+2} & a_{-n+1} & a_0 & \cdot & a_{-n+4} & a_{-n+3} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{-2} & a_{-3} & a_{-4} & \cdot & a_0 & a_{-1} \\ a_{-1} & a_{-2} & a_{-3} & \cdot & a_{-n+1} & a_0 \end{bmatrix}$$

Сумма циркулянтных матриц - циркулянтная матрица. Произведение двух циркулянтных матриц является циркулянтной матрицей.

Произведение любых двух циркулянтных матриц перестановочно.

Матрица, обратная к невырожденной циркулянтной матрице, является циркулянтной. Доказательства этих утверждений приведены в [1, стр. 41-45].

Матрица, элементы которой a_{ij} зависят только от суммы индексов $i + j$, называеся ***ганкелевой:***

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdot & a_{n-2} & a_{n-1} \\ a_1 & a_2 & a_3 & \cdot & a_{n-1} & a_n \\ a_2 & a_3 & a_4 & \cdot & a_n & a_{n+1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n-2} & a_{n-1} & a_n & \cdot & a_{2n-4} & a_{2n-3} \\ a_{n-1} & a_n & a_{n+1} & \cdot & a_{2n-3} & a_{2n-2} \end{bmatrix}$$

Ганкелева матрица является симметричной. В ганкелевой матрице равные элементы расположены на диагоналях, параллельных побочной. Ганкелева матрица полностью определяется первой строкой и последним столбцом.

Очевидно, что при перестановке строк (или столбцов) ганкелевой матрицы в обратном порядке получается теплицева матрица (и наоборот).

Решение систем линейных уравнений с циркулянтными, теплицевыми и ганкелевыми матрицами

Для матрицы, обратной к невырожденной теплицевой, известно несколько представлений в виде произведения двух теплицевых матриц специального вида [1, стр. 95; 2, стр. 120].

В частности, для матрицы, обратной к невырожденной теплицевой матрице A, справедливо следующее представление [1, стр.102]

$$A^{-1} = x_0^{-1}\left\{\begin{bmatrix} x_0 & 0 & 0 & . & 0 \\ x_1 & x_0 & 0 & . & 0 \\ . & . & . & . & . \\ x_{n-2} & x_{n-3} & x_{n-4} & . & 0 \\ x_{n-1} & x_{n-2} & x_{n-3} & . & x_0 \end{bmatrix}\begin{bmatrix} y_{n-1} & y_{n-2} & y_{n-3} & . & y_0 \\ 0 & y_{n-1} & y_{n-2} & . & y_1 \\ . & . & . & . & . \\ 0 & 0 & 0 & y_{n-1} & y_{n-2} \\ 0 & 0 & 0 & . & y_{n-1} \end{bmatrix}\right. \quad (1)$$

$$\left. - \begin{bmatrix} 0 & 0 & 0 & . & 0 \\ y_0 & 0 & 0 & . & 0 \\ . & . & . & . & . \\ y_{n-3} & y_{n-4} & y_{n-5} & . & 0 \\ y_{n-2} & y_{n-3} & y_{n-4} & . & 0 \end{bmatrix}\begin{bmatrix} 0 & x_{n-1} & x_{n-2} & . & x_1 \\ 0 & 0 & x_{n-1} & . & x_2 \\ . & . & . & . & . \\ 0 & 0 & 0 & 0 & x_{n-1} \\ 0 & 0 & 0 & . & 0 \end{bmatrix}\right\}$$

где $x=[x_0,x_1,..x_{n-1}]^T$ и $y=[y_0,y_1,...y_{n-1}]^T$ - решения систем линейных уравнений

$$Ax=[1,0,...0]^T \quad (2)$$

$$Ay=[0,...0,1]^T, \quad (3)$$

которые фактически определяют первую и последнюю строки матрицы A^{-1}.

Представление (1) справедливо при условии, что $x_0 \neq 0$.

Как следует из представления (1), матрица, обратная к теплицевой, определяется парой векторов x, y. Для вычисления матрицы, обратной к циркулянтной, достаточно найти только вектор x.

Наиболее быстрые алгоритмы обращения теплицевой матрицы порядка n требуют $O(n\log_2^2 n)$ арифметических операций, но являются эффективными лишь асимптотически при больших значениях n. При $n<1000$ они проигрывают алгоритмам с числом операций $O(n^2)$.

Ниже приведены рекуррентные формулы для вычисления векторов x и y, входящих в представление (1), которые требуют $O(n^2)$ арифметических операций. Алгоритм может применяться только для невырожденных теплицевых матриц, удовлетворяющих условию, что первая компонента x_0 вектора x, который является решением системы (2) , не равна 0.

Векторы x и y, входящие в представление (1), находятся из следующих рекуррентных формул [1, стр.149]:

$$x_0^{(0)} = y_0^{(0)} = 1/a_0$$
$$F_k = a_k x_0^{(k-1)} + a_{k-1}x_1^{(k-1)} + ... + a_1 x_{k-1}^{(k-1)}$$
$$G_k = a_{-1}y_0^{(k-1)} + a_{-2}y_1^{(k-1)} + ... + a_{-k}y_{k-1}^{(k-1)}$$

$$r_k = 1/(1-F_k G_k)$$
$$s_k = -r_k F_k$$
$$t_k = -r_k G_k$$

$$\left[x_0^{(k)}, x_1^{(k)}, \ldots x_k^{(k)}\right]^T = \left[x_0^{(k-1)}, x_1^{(k-1)}, \ldots x_{k-1}^{(k-1)}, 0\right]^T r_k + \left[0, y_0^{(k-1)}, y_1^{(k-1)}, \ldots y_{k-1}^{(k-1)}\right]^T s_k$$

$$\left[y_0^{(k)}, y_1^{(k)}, \ldots y_k^{(k)}\right]^T = \left[x_0^{(k-1)}, x_1^{(k-1)}, \ldots x_{k-1}^{(k-1)}, 0\right]^T t_k + \left[0, y_0^{(k-1)}, y_1^{(k-1)}, \ldots y_{k-1}^{(k-1)}\right]^T r_k$$

$k = 1, \ldots n-1$.

Если все ведущие миноры теплицевой матрицы отличны от нуля, то $1 - F_k G_k \neq 0$ для любого $k = 1, \ldots, n-1$ [1, стр.150].

Векторы x и y, входящие в представление (1) для обратной матрицы, получаем из рекуррентных формул при $k = n-1$:

$$x = \left[x_0, x_1, \ldots x_{n-1}\right]^T = \left[x_0^{(n-1)}, x_1^{(n-1)}, \ldots x_{n-1}^{(n-1)}\right]^T$$

$$y = \left[y_0, y_1, \ldots y_{n-1}\right]^T = \left[y_0^{(n-1)}, y_1^{(n-1)}, \ldots y_{n-1}^{(n-1)}\right]^T$$

При вычислении векторов x и y при каждом k необходимо выполнить $6k$ операций умножения и $4k$ операций сложения. Таким образом, число арифметических операций для вычисления векторов x и y в данном алгоритме равно $O\left(n^2\right)$. Число арифметических операций при вычислении обратной матрицы можно уменьшить, если учесть, что матрица, обратная к теплицевой, является персимметричной, т.е. симметричной относительно побочной диагонали [2, стр.122].

При вычислении векторов x и y для *симметричной теплицевой матрицы* число арифметических операций можно сократить вдвое, так как обратная матрица будет симметрична и персимметрична одновременно [1, стр. 154]. В этом случае

$x_i^{(k)} = y_{k-i}^{(k)}$, $\quad 0 \le i \le k$

$a_i = a_{-i}$, $\quad 0 \le i \le n-1$, откуда следует, что $F_k = G_k$ и $s_k = t_k$.

Задача решения системы линейных алгебраических уравнений с *ганкелевой матрицей* сводится к задаче для теплицевой матрицы, так как ганкелева матрица преобразуется к теплицевой простой перестановкой строк.

Литература.

1. В.В.Воеводин, Е.Е.Тыртышников. Вычислительные процессы с теплицевыми матрицами. - М.: Наука, 1987.
2. В.В.Воеводин, Ю.А.Кузнецов. Матрицы и вычисления. - М.: Наука, 1984

Оглавление

Учебное издание

БЕРЕЗИНА Нина Ивановна

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ «ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ: ЯЗЫК ПРОГРАММИРОВАНИЯ C#»

Учебное пособие

Издательский отдел Факультета вычислительной математики и кибернетики
МГУ имени М.В.Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы,
МГУ имени М.В.Ломоносова,2-й учебный корпус

Напечатано с готового оригинал-макета
Издательство ООО «МАКС Пресс»
Лицензия ИД N 00510 от 01.12.99 г. Подписано к печати 23.08.2010 г.
Формат 60x88 1/16. Усл.печ.л. 4,75. Тираж 100 экз. Заказ 361.

119992, ГСП-2, Москва, Ленинские горы,
МГУ им. М.В.Ломоносова, 2-й учебный корпус, 627 к.
Тел 939-3890, 939-3891. Тел./Факс 939-3891