**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.

26

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 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

∙ интерфейсa System.IComparable для сравнения объектов Edition по полю с  названием издания;

∙ интерфейсa 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 (), c 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

30

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 c 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.

**Код программы:**

1. Начало программы

class Program

{

static void NextTask()

{

Console.WriteLine("\n Нажмите любую клавишу для продолжения...");

Console.ReadKey();

Console.Clear();

}

static void Main(string[] args)

{

/\*

\* Создать объект типа MagazineCollection. Добавить в коллекцию несколько

элементов типа Magazine с разными значениями полей и вывести объект

MagazineCollection.

\*/

var magazineCollection = new MagazineCollection();

var firstMagazine = TestCollections.GenerateMagazine(1);

firstMagazine.ReleaseDate = new DateTime(2023, 4, 26);

firstMagazine.EditionCirculation = 5;

var secondMagazine = TestCollections.GenerateMagazine(2);

secondMagazine.ReleaseDate = new DateTime(2023, 2, 10);

secondMagazine.EditionCirculation = 10;

magazineCollection.AddMagazines(firstMagazine, secondMagazine);

Console.WriteLine(magazineCollection.ToShortString());

NextTask();

/\*

\* 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

Console.WriteLine("Сортировка по названию издания");

magazineCollection.SortByName();

Console.WriteLine(magazineCollection.ToShortString());

NextTask();

// по дате выхода издания

Console.WriteLine("Сортировка по дата выхода издания");

magazineCollection.SortByDate();

Console.WriteLine(magazineCollection.ToShortString());

NextTask();

// по тиражу издания

Console.WriteLine("Сортировка по тиражу издания");

magazineCollection.SortByEditionCirculation();

Console.WriteLine(magazineCollection.ToShortString());

NextTask();

/\*

\* 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;

Console.WriteLine("Максимального значения среднего рейтинга статей");

Console.WriteLine(magazineCollection.MaxAverageRating);

NextTask();

}

}

1. Класс Person – для описания человека

class Person

{

/\* Имя \*/

private string name;

/\* Фамилия \*/

private string secondName;

/\* Дата рождения \*/

private DateTime dateBirth;

/\*\* Имя \*/

public string Name

{

get { return name; }

}

/\* Фамилия \*/

public string SecondName

{

get { return secondName; }

}

/\* Дата рождения \*/

public DateTime DateBirth

{

get { return dateBirth; }

}

public int DateBirthYear

{

get { return dateBirth.Year; }

set { dateBirth.AddYears(value - dateBirth.Year); }

}

public Person(string name, string secondName, DateTime dateBirth)

{

this.name = name;

this.secondName = secondName;

this.dateBirth = dateBirth;

}

public Person()

{

name = "Егор";

secondName = "Дмитриев";

dateBirth = new DateTime(1999, 1, 1);

}

public override string ToString()

{

return $"Имя: {Name}\nФамилия: {SecondName}\nДата рождения: {DateBirth}";

}

public virtual string ToShortString()

{

return $"Имя: {Name}\nФамилия: {SecondName}";

}

public override bool Equals(object obj)

{

var person = obj as Person;

return name == person.name && secondName == person.secondName && dateBirth == person.dateBirth;

}

public static bool operator ==(Person obj1, Person obj2)

{

return obj1.Equals(obj2);

}

public static bool operator !=(Person obj1, Person obj2)

{

return !obj1.Equals(obj2);

}

public override int GetHashCode()

{

return ToString().GetHashCode();

}

public Person DeepCopy()

{

return new Person(Name, SecondName, DateBirth);

}

}

1. Класс Article – для описания статьи

class Article: IRateAndCopy

{

/\* Автор статьи \*/

public Person Person { get; set; }

/\* Название статьи \*/

public string Name { get; set; }

/\* Рейтинг статьи \*/

public double Rating { get; set; }

public Article(Person person, string name, double rating)

{

Person = person;

Name = name;

Rating = rating;

}

public Article()

{

Person = new Person();

Name = "Тестовая статья";

Rating = 2.1;

}

public override string ToString()

{

return $"Название статьи: {Name}\nРейтинг статьи: {Rating}\nАвтор статьи: {Person}";

}

public object DeepCopy()

{

return new Article(Person.DeepCopy(), Name, Rating);

}

}

1. Класс Edition – для описания издания

class Edition : IComparable, IComparable<Edition>

{

/\* Название издания \*/

protected string name;

/\* Дата выхода издания \*/

protected DateTime releaseDate;

/\* Тираж издания \*/

protected int editionCirculation;

/\* Название издания \*/

public string Name {

get { return name; }

set { name = value; }

}

/\* Дата выхода издания \*/

public DateTime ReleaseDate {

get { return releaseDate; }

set { releaseDate = value; }

}

/\* Тираж издания \*/

public int EditionCirculation {

get { return editionCirculation; }

set {

if(value < 0)

{

Exception exception = new Exception("EditionCirculation < 0");

throw exception;

}

editionCirculation = value;

}

}

public Edition(string name, DateTime releaseDate, int editionCirculation)

{

this.name = name;

this.releaseDate = releaseDate;

this.editionCirculation = editionCirculation;

}

public Edition()

{

name = "Тестовое издание";

releaseDate = new DateTime();

editionCirculation = 99;

}

public virtual object DeepCopy()

{

return new Edition(Name, releaseDate, editionCirculation);

}

public override bool Equals(object obj)

{

var edition = obj as Edition;

return name == edition.Name && releaseDate == edition.ReleaseDate && editionCirculation == edition.editionCirculation;

}

public override string ToString()

{

return $"Название издания: {name}\n" +

$"Дата выхода издания: {releaseDate}\n" +

$"Тираж издания: {editionCirculation}";

}

public override int GetHashCode()

{

return ToString().GetHashCode();

}

public int CompareTo(object o)

{

if (o is Edition edition) return Name.CompareTo(edition.Name);

else throw new ArgumentException("Некорректное значение параметра");

}

public int CompareTo(Edition edition)

{

return ReleaseDate.CompareTo(edition.ReleaseDate);

}

}

class EditionComparer : IComparer<Edition>

{

public int Compare(Edition x, Edition y)

{

return x.ReleaseDate.CompareTo(y.ReleaseDate);

}

}

1. Класс Magazine – для описания журнала

class Magazine : Edition, IRateAndCopy

{

/\* Переодичность выхода журнала \*/

private EnumFrequency qutputFrequency;

/\* Редакторы журнала \*/

private List<Person> editors;

/\* Cписок статей в журнале \*/

private List<Article> articles;

/\* Переодичность выхода журнала \*/

public EnumFrequency QutputFrequency

{

get { return qutputFrequency; }

set { qutputFrequency = value; }

}

/\* Статьи в журнале \*/

public List<Article> Articles

{

get { return articles; }

set { articles = value; }

}

/\* Редакторы журнала \*/

public List<Person> Editors

{

get { return editors; }

set { editors = value; }

}

/\* Cреднее значение рейтинга в списке статей \*/

public double AvgRatingOfArticles

{

get

{

return articles.Sum(a => a.Rating) / articles.Count();

}

}

public bool this[EnumFrequency frequency]

{

get => QutputFrequency == frequency;

}

/\* Получаем строковое представление частоты выхода журнала \*/

private string QutputFrequencyString

{

get

{

switch (qutputFrequency)

{

case EnumFrequency.Weekly:

return "Ежедневно";

case EnumFrequency.Monthly:

return "Ежемесячно";

case EnumFrequency.Yearly:

return "Ежегодно";

default:

return "";

}

}

}

/\* Получаем строку с информацией о статьях \*/

private string ArticlesInfoString

{

get {

return articles.Select((article) => article.ToString()).Aggregate((current, next) => current + "\n\n" + next);

}

}

/\* Строка с информацие о редакторах \*/

private string EditorsInfoString

{

get

{

return editors.Select((editor) => editor.ToString()).Aggregate((current, next) => current + "\n\n" + next);

}

}

/\* Рейтинг всех журналов \*/

public double Rating

{

get

{

return articles.Sum((article) => article.Rating);

}

}

/\* Данные издателя \*/

public Edition Edition

{

get

{

return new Edition(Name, ReleaseDate, EditionCirculation);

}

set

{

Name = value.Name;

ReleaseDate = value.ReleaseDate;

EditionCirculation = value.EditionCirculation;

}

}

public IEnumerable<Article> GetArticlesForRating(double min = 0) {

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

{

var article = Articles[i];

if(article.Rating > min)

{

yield return article;

}

}

}

public IEnumerable<Article> GetArticlesForSubstr(string substring)

{

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

{

var article = Articles[i];

if (article.Name.Contains(substring))

{

yield return article;

}

}

}

public Magazine(string name, EnumFrequency qutputFrequency, DateTime releaseDate, int circulation, List<Article> articles, List<Person> editors)

{

this.name = name;

this.qutputFrequency = qutputFrequency;

this.releaseDate = releaseDate;

editionCirculation = circulation;

this.articles = articles;

this.editors = editors;

}

public Magazine()

{

name = "Тестовый журнал";

qutputFrequency = EnumFrequency.Monthly;

releaseDate = DateTime.MinValue;

editionCirculation = 0;

articles = new List<Article>() { new Article(), new Article() };

editors = new List<Person>() { new Person(), new Person() };

}

/\* Добавляем статьи в журнал \*/

public void AddArticles(params Article[] newArticles)

{

articles = articles.Concat(newArticles).ToList();

}

/\* Добавляем редакторов в журнал \*/

public void AddEditors(params Person[] newEditors)

{

editors = editors.Concat(newEditors).ToList();

}

public override string ToString()

{

return $"Название журнала: {name}\n" +

$"Периодичность выхода журнала: {QutputFrequencyString}\n" +

$"Дата выхода журнала: {releaseDate}\n" +

$"Тираж: {editionCirculation}\n\n" +

$"Статьи:\n" +

$"{ArticlesInfoString}\n\n" +

$"Редакторы:\n" +

$"{EditorsInfoString}";

}

public virtual string ToShortString()

{

return $"Название журнала: {name}\n" +

$"Периодичность выхода журнала: {QutputFrequencyString}\n" +

$"Дата выхода журнала: {releaseDate}\n" +

$"Тираж: {editionCirculation}\n" +

$"Cреднее значение рейтинга статей: {AvgRatingOfArticles}";

}

public override object DeepCopy()

{

return new Magazine(Name, QutputFrequency, ReleaseDate, EditionCirculation, Articles, Editors);

}

}

1. Класс MagazineCollection – для описания коллекции журналов

// Класс, реализующий интерфейс IComparer<Edition> для сортировки по дате выхода

class EditionDateComparer : IComparer<Magazine>

{

public int Compare(Magazine x, Magazine y)

{

return x.ReleaseDate.CompareTo(y.ReleaseDate);

}

}

// Класс, реализующий интерфейс IComparer<Edition> для сортировки по тиражу

class EditionyEditionCirculationComparer : IComparer<Magazine>

{

public int Compare(Magazine x, Magazine y)

{

return x.EditionCirculation.CompareTo(y.EditionCirculation);

}

}

class NameComparer: IComparer<Magazine>

{

public int Compare(Magazine x, Magazine y)

{

return x.Name.CompareTo(y.Name);

}

}

class MagazineCollection

{

/\*

\* Список журналов

\*/

private List<Magazine> magazines = new List<Magazine>();

// Cвойство, возвращающее максимальное значение среднего рейтинга статей

public double MaxAverageRating

{

get

{

if (magazines == null || magazines.Count == 0) return 0;

return magazines.Max(m => m.AvgRatingOfArticles);

}

}

// Cвойство, возвращающее подмножество элементов с периодичностью выхода журнала Monthly

public IEnumerable<Magazine> MonthlyMagazines

{

get {

return magazines.Where(m => m.QutputFrequency == EnumFrequency.Monthly);

}

}

// Метод, возвращающий список элементов со средним рейтингом статей больше или равным заданному значению

public List<Magazine> RatingGroup(double value)

{

return magazines.GroupBy(m => m.AvgRatingOfArticles >= value).Where(g => g.Key).SelectMany(g => g).ToList();

}

/\*

\* Добавляем журналы по умолчанию

\*/

public void AddDefault()

{

var article1 = new Article(new Person("Иван", "Федоров", new DateTime(2000, 07, 18)), "Статья №1", 5);

var article2 = new Article(new Person("Петров", "Сергеевич", new DateTime(2000, 10, 12)), "Статья №2", 10);

var article3 = new Article(new Person("Александр", "Куприлович", new DateTime(1995, 05, 27)), "Статья №3", 23);

magazines.Add(new Magazine("Журнал №1", EnumFrequency.Yearly, new DateTime(2020, 09,20), 500, new List<Article> { article1, article2, article3 }, new List<Person> { new Person("Сергей", "Иванов", new DateTime(1999, 10, 14))}));

magazines.Add(new Magazine("Журнал №2", EnumFrequency.Yearly, new DateTime(2020, 09,20), 500, new List<Article> { article1, article3 }, new List<Person> { new Person("Сергей1", "Иванов1", new DateTime(1999, 10, 14))}));

}

/\*

\* Добавить журналы

\*/

public void AddMagazines(params Magazine[] \_magazines)

{

magazines.AddRange(\_magazines);

}

// Сортировка по названию издания

public void SortByName()

{

magazines.Sort(new NameComparer());

}

// Сортировка по дате выхода издания

public void SortByDate()

{

magazines.Sort(new EditionDateComparer());

}

// Сортировка по тиражу издания

public void SortByEditionCirculation()

{

magazines.Sort(new EditionyEditionCirculationComparer());

}

public override string ToString()

{

return magazines.Aggregate("", (acc, next) => acc + "\n===\n" + next.ToString());

}

public virtual string ToShortString()

{

return magazines.Aggregate("", (acc, next) => acc + "\n===\n" + next.ToShortString());

}

}

1. Класс TestCollections – для тестирования и заполнения коллекций

class TestCollections

{

/\* Издатели \*/

private List<Edition> editions;

/\* Названия издателей \*/

private List<string> editionNames;

/\* Журналы по ключу издателя \*/

private Dictionary<Edition, Magazine> editionMagazine;

/\* Журналы по ключу имени издателя \*/

private Dictionary<string, Magazine> editionNameMagazine;

/\* Генерируем объект Person \*/

public static Person GeneratePerson(int i)

{

return new Person($"PersonName - {i}", $"PersonSecondName - i", new DateTime());

}

/\* Генерируем объект Article \*/

public static Article GenerateArticle(int i)

{

return new Article(GeneratePerson(i), $"Article - {i}", i \* 5);

}

/\*

\* Генерируем объект класс Magazine

\*/

public static Magazine GenerateMagazine(int i)

{

/\* Список редакторов \*/

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

/\* Заполняем список редакторов случайными значениями \*/

for (int j = 0; j < 2; j++)

editors.Add(GeneratePerson(i + j));

/\* Список статей \*/

List<Article> articles = new List<Article>();

/\* Заполняе список статеть случайными значениями \*/

for (int j = 0; j < 2; j++)

articles.Add(GenerateArticle(i + j));

Random random = new Random();

/\* Генерируем рандомное значение enum \*/

EnumFrequency frequency = (EnumFrequency)Enum.GetValues(typeof(EnumFrequency)).GetValue(random.Next(1, 3));

/\* Создаем и возвращаем объект Magazine \*/

return new Magazine($"Magazine - {i}", frequency, new DateTime(), i \* 28, articles, editors);

}

public TestCollections(int count)

{

/\* Издатели \*/

editions = new List<Edition>();

/\* Названия издателей \*/

editionNames = new List<string>();

/\* Журналы по издателю \*/

editionMagazine = new Dictionary<Edition, Magazine>();

/\* Журналы по имени издателя \*/

editionNameMagazine = new Dictionary<string, Magazine>();

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

{

/\* Генерируем журнал \*/

var magazine = GenerateMagazine(i);

/\* Добавляем издателя \*/

editions.Add(magazine.Edition);

/\* Добавляем имя издателя \*/

editionNames.Add(magazine.Name);

/\* Добавляем журнал по издателю \*/

editionMagazine.Add(magazine.Edition, magazine);

/\* Добавляем журналы по имени издателя \*/

editionNameMagazine.Add(magazine.Name, magazine);

}

}

public void Test(string editionName)

{

// Test find List<string>

int startTimer = Environment.TickCount;

string edition = editionNames.Find((item) => item == editionName);

Console.WriteLine($"Test find List<string> - {Environment.TickCount - startTimer}");

// Test find List<Edition>

startTimer = Environment.TickCount;

Edition edition1 = editions.Find((item) => item.Name == editionName);

Console.WriteLine($"Test find List<Edition> - {Environment.TickCount - startTimer}");

if(edition1 != null)

{

// Test find Dictionary<Edition, Magazine>

startTimer = Environment.TickCount;

Magazine magazine = editionMagazine[edition1];

Console.WriteLine($"Test find Dictionary<Edition, Magazine> - {Environment.TickCount - startTimer}");

}

// Test find Dictionary<string, Magazine>

startTimer = Environment.TickCount;

Magazine magazine1 = editionNameMagazine[editionName];

Console.WriteLine($"Test find Dictionary<string, Magazine> - {Environment.TickCount - startTimer}");

}

}

**Консольный вывод программы:**









