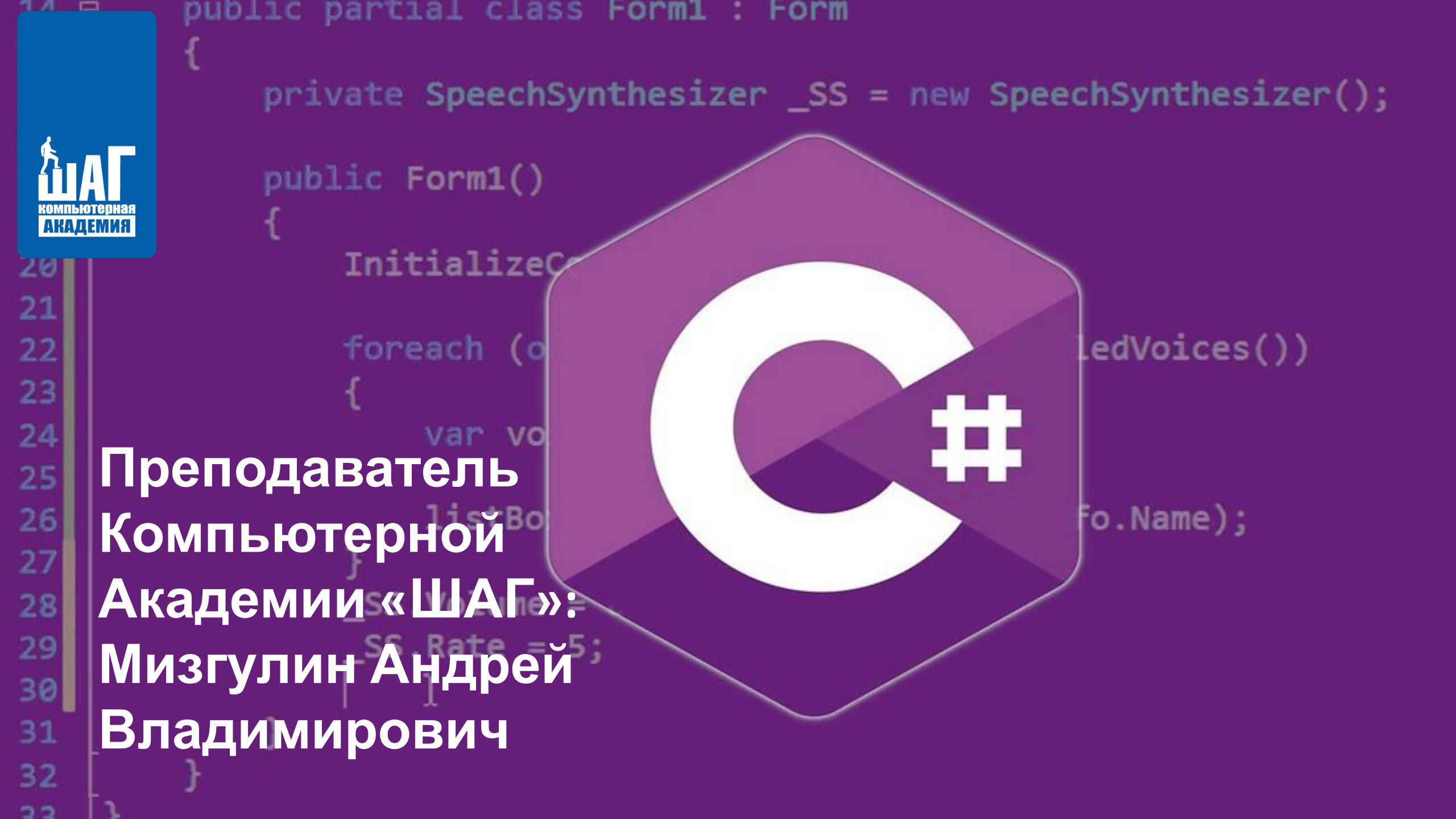
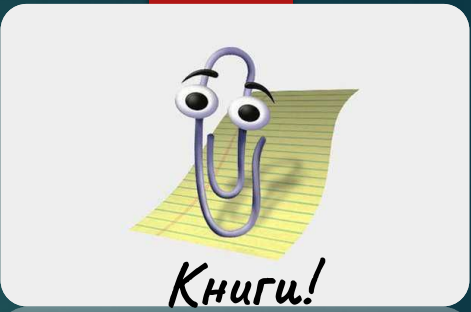

ШАГ
компьютерная
АКАДЕМИЯ
Преподаватель Компьютерной Академии «ШАГ»: Мизгулин Андрей Владимирович

2

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Референсы!



https://drive.google.com/drive/folders/1YO2Bq5xYDYo5VPyYTPTq0gyG4RyTH5oh?usp=sharing

3

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Референсы!

YouTube

Видео!

Создание игры “Space Invaders” на C# с нуля
https://youtu.be/Zw8DVvmutGc

Как стать C# разработчиком в 2021 году. .NET или .NET Core?
https://youtu.be/z8XrqlOcjD0

!

Уроки C# (C sharp) | #1 - Что такое C# и зачем он нужен?
https://youtu.be/3FWqP80fNJM

C# ОТ НОВИЧКА К ПРОФЕССИОНАЛУ
https://youtu.be/KyFWqbRfWIA

ИНТЕРЕСНО!!!

4

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Референсы!

Полезные материалы по языку от Microsoft и уроки для начинающих C#-программистов.
https://tproger.ru/tag/c-sharp/

Поддержка нескольких репозиториев в Visual Studio
https://habr.com/ru/company/microsoft/blog/573426/

Статьи по C# и .NET
https://metanit.com/sharp/articles/

Документация по C#
https://docs.microsoft.com/ru-ru/dotnet/csharp/

Программирование на C, C# и Java
https://vscode.ru/category/articles

5

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Экзамен!

Задание 1

Создать приложение «Словари».

Основная задача проекта: хранить словари на разных языках и разрешать пользователю находить перевод нужного слова или фразы.

Интерфейс приложения должен предоставлять такие возможности:

- Создавать словарь. При создании нужно указать тип словаря. Например, англо-русский или русско-английский.
- Добавлять слово и его перевод в уже существующий словарь. Так как у слова может быть несколько переводов, необходимо поддерживать возможность создания нескольких вариантов перевода.
- Заменять слово или его перевод в словаре.
- Удалять слово или перевод. Если удаляется слово, все его переводы удаляются вместе с ним. Нельзя удалить перевод слова, если это последний вариант перевода.
- Искать перевод слова.
- Словари должны храниться в файлах.
- Слово и варианты его переводов можно экспортировать в отдельный файл результата.
- При старте программы необходимо показывать меню для работы с программой. Если выбор пункта меню открывает подменю, то тогда в нем требуется предусмотреть возможность возврата в предыдущее меню.

Задание 2

Создать приложение «Викторина».

Основная задача проекта: предоставить пользователю возможность проверить свои знания в разных областях.

Интерфейс приложения должен предоставлять такие возможности:

- При старте приложения пользователь вводит логин и пароль для входа. Если пользователь не зарегистрирован, он должен пройти процесс регистрации.
- При регистрации нужно указать:
  - логин (нельзя зарегистрировать уже существующий логин);
  - пароль;
  - дату рождения.
- После входа в систему пользователь может:
  - стартовать новую викторину;
  - посмотреть результаты своих прошлых викторин;
  - посмотреть Топ-20 по конкретной викторине;
  - изменить настройки. Можно менять пароль и дату рождения;
  - выход.
- Для старта новой викторины пользователь должен выбрать раздел знаний викторины. Например: «История», “География», «Биология» и т. д. Также нужно предусмотреть смешанную викторину, когда вопросы будут выбираться из разных викторин по случайному принципу.
- Конкретная викторина состоит из двадцати вопросов. У каждого вопроса может быть один или несколько правильных вариантов ответа. Если вопрос предполагает несколько правильных ответов, а пользователь указал не все, вопрос не засчитывается.
- По завершении викторины пользователь получает количество правильно отвеченных вопросов, а также свое место в таблице результатов игроков викторины.

Необходимо также разработать утилиту для создания и редактирования викторин и их вопросов. Это приложение должно предусматривать вход по логину и паролю.

6

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Экзамен!

Задание 3

Создать приложение ИС «Агентство недвижимости».

Основная задача проекта: хранить информацию об объектах недвижимости и выбирать эти объекты по критериям выборки заданный пользователем.

Интерфейс приложения должен предоставлять такие возможности:

- Создать объект недвижимости, задать ему характеристики (адрес, площадь, кадастровая цена);
- Редактировать уже существующий объект недвижимости;
- Вход в программу должен осуществляться по логину и пароля;
- В программе должны быть выделены роли пользователей (агент, пользователь, администратор);
- Осуществлять поиск объекта недвижимости по любому критерию;
- Информация об объектах недвижимости должна храниться в файлах;
- Информацию по конкретному объекту недвижимости должна иметь возможность быть экспортирована в отдельный файл;
- При старте программы необходимо показывать меню для работы с программой. Если выбор пункта меню открывает подменю, то тогда в нем требуется предусмотреть возможность возврата в предыдущее меню;
- Показ некоторых меню должен завысить от роли под которой произошел вход в программу. Например: пользователю, должно быть невозможно создавать/редактировать объект недвижимости. Агенту должно быть невозможно добавлять пользователей. Администратору должны быть доступны все функции.

Задание 4

Создать приложение ИС «Регистрация автотранспорта».

Основная задача проекта: хранить информацию о транспортных средствах.

Интерфейс приложения должен предоставлять такие возможности:

- При старте приложения пользователь вводит логин и пароль для входа. Если пользователь не зарегистрирован, он должен пройти процесс регистрации.
- При регистрации нужно указать:
  - логин (нельзя зарегистрировать уже существующий логин);
  - пароль;
  - дату рождения.
- После входа в систему пользователь может:
  - зарегистрировать транспортное средство, указав модель, дату выпуска, характеристики, номер двигателя и т.д.;
  - посмотреть историю по владельцам транспортного средства;
  - экспортировать в отдельный файл информацию по конкретному транспортному средству;
  - изменять информацию по транспортному средству;
  - найти транспортное средство по любому критерию;
  - выход.
- Информация по транспортным средствам должна храниться в файлах;
- В приложении обязательно должен вестись лог работы:
  - в логе должны быть записаны дата и время входа конкретных пользователей, а также все действия, которые производят эти пользователи.

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Референсы!

Перегрузка операторов

8

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программиро

Введение
в перегрузку операторов!

Перегрузка операторов представляет собой механизм определения стандартных операций для пользовательских типов. Для встроенных типов языка C# перегрузка стандартных операторов уже реализована, и ее изменить невозможно. Например, в классе System.String перегрузка операторов == и != используется для проверки равенства и неравенства строк по их содержимому, а оператор + перегружен и выполняет конкатенацию строк. Хочется обратить Ваше внимание, что перегрузка операторов является одним из способов полиморфизма, например, применяя операцию + к числовым типам, мы получим значение их суммы, а применяя ее же со строками, получим конкатенацию строк.

Перегружаемые операторы

Следующая таблица содержит сведения о возможности перегрузки операторов C#:

Операторы	Возможность перегрузки
+x, -x, !x, ~x, ++, --, true, false	Эти унарные операторы могут быть перегружены.
x + y, x - y, x * y, x / y, x % y, x & y, x | y, x ^ y, x << y, x >> y, x == y, x != y, x < y, x > y, x <= y, x >= y	Эти бинарные операторы могут быть перегружены. Некоторые операторы должны перегружаться парами; дополнительные сведения см. в примечаниях после этой таблицы.
x && y, x || y	Условный логический оператор не может быть перегружен. При этом, если тип с перегруженными операторами true и false также перегружает оператор `&` или `|`, оператор `&&` или `||`, соответственно, может быть применен для операндов этого типа. Дополнительные сведения см. в разделе Пользовательские условные логические операторы в Спецификации языка C#.
a[i], a?[i]	Доступ к элементам не считается перегружаемым оператором, но вы можете определить индексатор.
(T)x	Оператор приведения невозможно перегрузить, но можно определить пользовательские преобразования типа и выполнять его с помощью выражения приведения. Дополнительные сведения см. в разделе Операторы пользовательского преобразования.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Составные операторы присваивания не могут быть перегружены явным образом. Однако при перегрузке бинарного оператора соответствующий составной оператор присваивания (если таковой имеется) также неявно перегружается. Например, `+=` вычисляется с помощью `+`, который может быть перегружен.
^x, x = y, x.y, x?.y, c ? t : f, x ?? y, x ??= y, x..y, x->y, =>, f(x), as, await, checked, unchecked, default, delegate, is, nameof, new, sizeof, stackalloc, switch, typeof, with	Эти операторы не могут быть перегружены.

9

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программиро

Введение
в перегрузку операторов!

Перегрузка операторов используется для улучшения читабельности программ и должна соответствовать определенным требованиям:

- **перегрузка операторов должна выполняться открытыми статическими методами класса;**
- **у метода-оператора тип возвращаемого значения или одного из параметров должен совпадать с типом, в котором выполняется перегрузка оператора;**
- **параметры метода-оператора не должны включать модификатор out и ref.**

Перегружаемые операторы

Следующая таблица содержит сведения о возможности перегрузки операторов C#:

Операторы	Возможность перегрузки
+x, -x, !x, ~x, ++, --, true, false	Эти унарные операторы могут быть перегружены.
x + y, x - y, x * y, x / y, x % y, x & y, x | y, x ^ y, x << y, x >> y, x == y, x != y, x < y, x > y, x <= y, x >= y	Эти бинарные операторы могут быть перегружены. Некоторые операторы должны перегружаться парами; дополнительные сведения см. в примечаниях после этой таблицы.
x && y, x || y	Условный логический оператор не может быть перегружен. При этом, если тип с перегруженными операторами true и false также перегружает оператор `&` или `|`, оператор `&&` или `||`, соответственно, может быть применен для операндов этого типа. Дополнительные сведения см. в разделе Пользовательские условные логические операторы в Спецификации языка C#.
a[i], a?[i]	Доступ к элементам не считается перегружаемым оператором, но вы можете определить индексатор.
(T)x	Оператор приведения невозможно перегрузить, но можно определить пользовательские преобразования типа и выполнять его с помощью выражения приведения. Дополнительные сведения см. в разделе Операторы пользовательского преобразования.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Составные операторы присваивания не могут быть перегружены явным образом. Однако при перегрузке бинарного оператора соответствующий составной оператор присваивания (если таковой имеется) также неявно перегружается. Например, `+=` вычисляется с помощью `+`, который может быть перегружен.
^x, x = y, x.y, x?.y, c ? t : f, x ?? y, x ??= y, x.y, x->y, =>, f(x), as, await, checked, unchecked, default, delegate, is, nameof, new, sizeof, stackalloc, switch, typeof, with	Эти операторы не могут быть перегружены.

10

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка
унарных операторов

```
public static тип_возврата
              operator символ_операции(параметры)
{
// код
}
```

Синтаксис перегрузки выглядит следующим образом. Поскольку перегруженные операторы являются статическими методами, они не получают указателя this, поэтому унарные операторы должны получать единственный операнд. Этот операнд должен иметь тип класса, в котором выполняется перегрузка оператора. То есть, если выполняется перегрузка унарного оператора в классе Point, то и тип операнда должен быть Point. Рассмотрим перегрузку унарных операторов на примере операторов инкремента, декремента и изменения знака –. Для операторов ++ и -- возвращаемое значение должно быть того же типа, в котором выполняется перегрузка оператора или производного от него. Класс Point описывает точку на плоскости с координатами x и y. Оператор инкремента увеличивает обе координаты на 1, оператор декремента соответственно уменьшает на 1, оператор — изменяет знак координат на противоположный.

11

ШАГ компьютерная АКАДЕМИЯ

Платформа Micr... я... программирован...

Перегрузка
унарных операторов

В данном примере Point является ссылочным типом, поэтому изменения значений x и y, которые выполняются в перегруженных операторах инкремента и декремента, изменяют переданный в них объект. Оператор — (изменение знака) не должен изменять состояние переданного объекта, а должен возвращать новый объект с измененным знаком. Для этого в реализации этого метода создается новый объект Point, изменяется знак его координат и этот объект возвращается из метода.

```
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
        //перегрузка инкремента
        public static Point operator ++(Point
s)
        {
            s.X++;
            s.Y++;
            return s;
        }
        //перегрузка декремента
        public static Point operator --(Point s)
        {
            s.X--;
            s.Y--;
            return s;
        }
        //перегрузка оператора -
        public static Point operator -(Point s)
        {
            return new Point { X = -s.X, Y =
-s.Y };
        }
        public override string ToString()
        {
            return $"Point: X = {X}, Y = {Y}";
        }
    }
```

```
 class Program
    {
        static void Main()
        {
            Point point = new Point { X =
10, Y = 10 };
            WriteLine($"Исходная
точка\n{point}");
            WriteLine("Префиксная и
постфиксная формы инкремента
выполняются одинаково");
            WriteLine(++point); // x=11,
y=11
            WriteLine(point++); // x=12,
y=12
            WriteLine($"Префиксная
форма декремента\n { --point} ");
            WriteLine($"Выполнение
оператора –\n { -point}");
            WriteLine($"не изменило
исходную точку\n { point}");
        }
    }
}
```

12

ШАГ компьютерная АКАДЕМИЯ

Платформа Micr… … я программирован…

Перегрузка
унарных операторов

Интересно отметить, что с C# нет возможности выполнить отдельно перегрузку постфиксной и префиксной форм операторов инкремента и декремента. Поэтому при вызове постфиксная и префиксная форма работают одинаково, как префиксная форма.

```
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
        //перегрузка инкремента
        public static Point operator ++(Point
s)
        {
            s.X++;
            s.Y++;
            return s;
        }
        //перегрузка декремента
        public static Point operator --(Point s)
        {
            s.X--;
            s.Y--;
            return s;
        }
        //перегрузка оператора -
        public static Point operator -(Point s)
        {
            return new Point { X = -s.X, Y =
-s.Y };
        }
        public override string ToString()
        {
            return $"Point: X = {X}, Y = {Y}";
        }
    }
```

```
 class Program
    {
        static void Main()
        {
            Point point = new Point { X =
10, Y = 10 };
            WriteLine($"Исходная
точка\n{point}");
            WriteLine("Префиксная и
постфиксная формы инкремента
выполняются одинаково");
            WriteLine(++point); // x=11,
y=11
            WriteLine(point++); // x=12,
y=12
            WriteLine($"Префиксная
форма декремента\n { --point} ");
            WriteLine($"Выполнение
оператора –\n { -point}");
            WriteLine($"не изменило
исходную точку\n { point}");
        }
    }
}
```

13

Платформа Microsoft и C# (шарп) я программирования.

Перегрузка бинарных операторов

Как отмечалось ранее, перегруженные операторы являются статическими методами, поэтому бинарные операторы должны получать два параметра. Для примера перегрузки бинарных операций освежим некоторые знания за 8 класс общеобразовательной школы — векторы.

Итак, вектор — направленный отрезок, имеющий начало и конец, то есть две точки. Для того чтобы получить координаты вектора необходимо из координат конечной точки вычесть соответствующие координаты начальной точки. Для того чтобы сложить два вектора нужно сложить их соответствующие координаты, разность — аналогично. Для того чтобы умножить вектор на число, необходимо каждую координату вектора умножить на это число. Создадим класс Vector, используя разработанный ранее класс Point.

```
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
    }
    class Vector
    {
        public int X { get; set; }
        public int Y { get; set; }
        public Vector() { }
        public Vector(Point begin, Point end)
        {
            X = end.X - begin.X;
            Y = end.Y - begin.Y;
        }
        public static Vector operator +(Vector v1,
        Vector v2)
        {
            return new Vector
            {
                X = v1.X + v2.X,
                Y = v1.Y + v2.Y
            };
        }
        public static Vector operator -(Vector v1,
        Vector v2)
        {
            return new Vector
            {
                X = v1.X - v2.X,
                Y = v1.Y - v2.Y
            };
        }
        public static Vector operator *(Vector v, int n)
        {
            v.X *= n;
            v.Y *= n;
            return v;
        }
```

Платформа Microsoft и C# (шарп) я программирования.

Перегрузка бинарных операторов

Перегрузка операторов +=, *=, -= выполняется автоматически после перегрузки соответствующих бинарных операторов, поэтому применение операции *= в коде ошибки не вызовет, а будет использован перегруженный оператор *.
Однако, перегруженные в примере, операторы будут использоваться компилятором, только если переменная типа Vector находится слева от знака операнда. То есть выражение v1*10 откомпилируется нормально, а при перестановке сомножителей — в выражении 10*v1 произойдет ошибка на этапе компиляции. Для исправления этой ошибки следует перегрузить оператор * с другим порядком операндов:

```
public static Vector operator *(Vector v, int n)
        {
            return v * n;
        }
```

```
public override string ToString()
        {
            return $"Vector: X = {X}, Y = {Y}";
        }
    }
    class Program
    {
        static void Main()
        {
            Point p1 = new Point { X = 2, Y = 3
};
            Point p2 = new Point { X = 3, Y = 1
};
            Vector v1 = new Vector(p1, p2);
            Vector v2 = new Vector { X = 2, Y =
3 };

WriteLine($"\tВектора\n{v1}\n{v2}");
            WriteLine($"\n\tСложение
векторов\n{v1 + v2}\n"); // x=3, y=1
            WriteLine($"\tРазность
векторов\n{v1 -  v2}\n"); // x=-1, y=-5
            WriteLine("Введите целое
число");
            int n = int.Parse(ReadLine());
            v1 *= n;
            WriteLine($"\n\tУмножение
вектора на число { n}\n{v1}\n ");
        }
    }
}
```

15

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов отношений

Операции сравнения перегружаются парами: если перегружается операция ==, также должна перегружаться операция !=. Существуют следующие пары операторов сравнения:

- **== и !=**
- **< и >**
- **<= и >=.**

При перегрузке операторов отношения надо учитывать, что есть два способа проверки равенства:

- **равенство ссылок (тождество);**
- **равенство значений.**

В классе Object определены следующие методы сравнения объектов:

- **public static bool ReferenceEquals(Object obj1, Object obj2)**
- **public bool virtual Equals(Object obj)**

Есть отличия в работе этих методов со значимыми и ссылочными типами.

Платформа Microsoft и C# (шарп) я программирования.

Перегрузка операторов отношений

Метод ReferenceEquals() проверяет, указывают ли две ссылки на один и тот же экземпляр класса; точнее — содержат ли две ссылки одинаковый адрес памяти. Этот метод невозможно переопределить. Со значимыми типами ReferenceEquals() всегда возвращает false, т.к. при сравнении выполняется приведение к Object и упаковка, упакованные объекты располагаются по разным адресам. Метод Equals() является виртуальным. Его реализация в Object выполняет проверку равенства ссылок, т.е. работает, так же как и ReferenceEquals(). Для значимых типов в базовом типе System.ValueType выполнено переопределение метода Equals(), в котором выполняется сравнение объектов путем сравнения всех полей (побитовое сравнение). Пример использования методов ReferenceEquals() и Equals() со ссылочными и значимыми типами:

```
using static System.Console;
namespace SimpleProject
{
    class CPoint
    {
        public int X { get; set; }
        public int Y { get; set; }
    }
    struct SPoint
    {
        public int X { get; set; }
        public int Y { get; set; }
    }
    class Program
    {
        static void Main()
        {
            // работа метода ReferenceEquals
со
            // ссылочными и значимыми
типами
            //ссылочный тип
            CPoint cp = new CPoint { X = 10, Y
= 10 };
            CPoint cp1 = new CPoint { X = 10,
Y = 10 };
            CPoint cp2 = cp1;
            // хотя p и p1 содержат
одинаковые
            // значения, они указывают на
разные
            // адреса памяти
            WriteLine($"ReferenceEquals(cp,
cp1) =  { ReferenceEquals(cp, cp1)} "); //
false
```

17

Платформа Microsoft и C# (шарп) я программирования

Перегрузка операторов отношений

```
Консоль отладки Microsoft Visual Studio
ReferenceEquals(cp, cp1) =  False
ReferenceEquals(cp1, cp2) =  True
ReferenceEquals(sp, sp) =  False
Equals(cp, cp1) =  False
Equals(sp, sp1) =  True
```

Перегрузка оператора == обычно выполняется путем вызова метода Equals(), поэтому необходимо переопределить метод Equals() в собственном типе. Поскольку в System.ValueType метод Equals() выполняет побитовое сравнение, то в собственных значимых типах его можно и не переопределять. Однако, в System.ValueType получение значений полей для сравнения в методе Equals() выполняется с помощью рефлексии, что приводит к снижению производительности. Поэтому при разработке значимого типа для увеличения быстродействия все-таки рекомендуется выполнить переопределение метода Equals().

```
// p1 и p2 указывают на один и тот же
            // адрес памяти
            WriteLine($"ReferenceEquals(cp1,
cp2) =  { ReferenceEquals(cp1, cp2)} "); //
true
            // значимый тип
            SPoint sp = new SPoint { X = 10, Y
= 10 };
            // при передаче в метод
ReferenceEquals
            // выполняется упаковка,
упакованные
            // объекты располагаются
            // по разным адресам
            WriteLine($"ReferenceEquals(sp,
sp) =  { ReferenceEquals(sp, sp)} "); //
false
            // работа метода Equals со
ссылочными
            // и значимыми типами
            // выполняется сравнение
адресов
            // ссылочных типов
            WriteLine($"Equals(cp, cp1) =  {
Equals(cp, cp1)} "); // false
            // значимый тип
            SPoint sp1 = new SPoint { X = 10,
Y = 10 };
            // выполняется сравнение
значений полей
            WriteLine($"Equals(sp, sp1) =  {
Equals(sp, sp1)} "); // true
        }
    }
}
```

Платформа Microsoft и C# (шарп) я программирования.

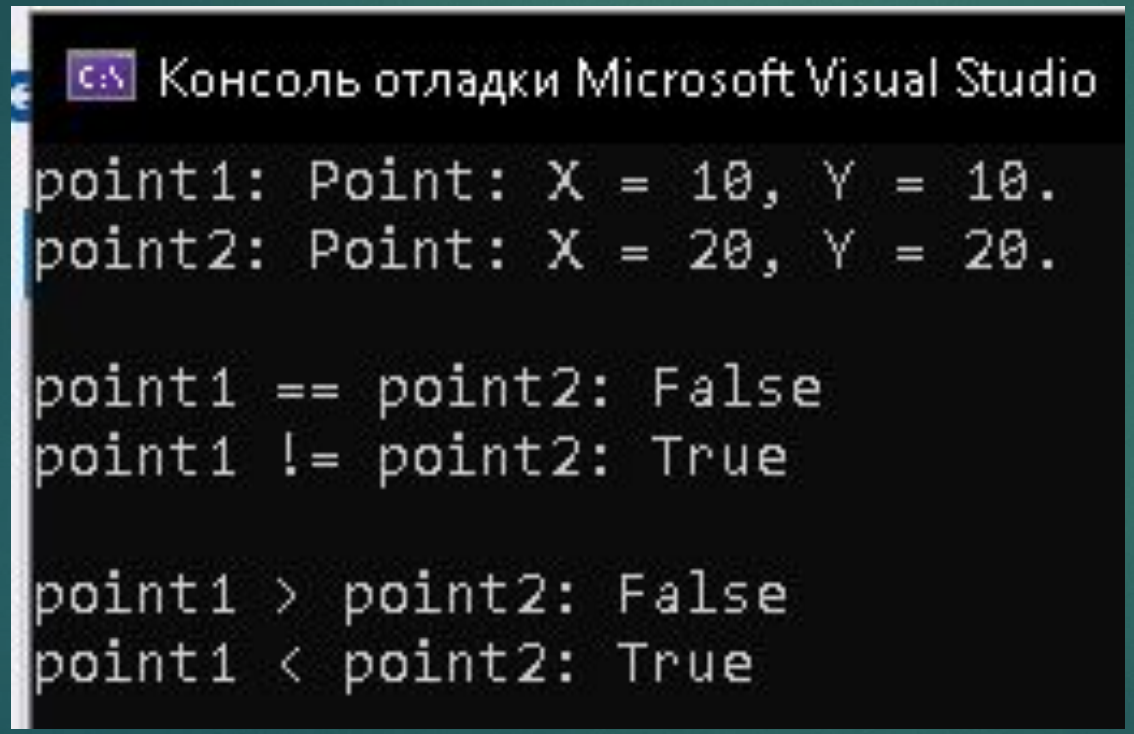
Перегрузка операторов отношений

При переопределении метода Equals() следует также переопределять и метод GetHashCode(). Этот метод предназначен для получения целочисленного значения хеш-кода объекта, при этом различным объектам должны соответствовать различные хеш–коды. Если перегрузку метода GetHashCode() не выполнить, то компилятор выдаст предупреждение. При перегрузке оператора != мы воспользовались оператором логического отрицания (!) и перегруженным оператором ==. в качестве сравнения двух точек при перегрузке операторов <и> было взято расстояние между заданной точкой и точкой с координатами (0, 0), которое вычисляется с использованием теоремы Пифагора. Пример перегрузки операторов отношений для класса Point:

```
using System;
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
        // переопределение метода Equals
        public override bool Equals(object obj)
        {
            return this.ToString() == obj.ToString();
        }
        // необходимо также переопределить метод
        // GetHashCode
        public override int GetHashCode()
        {
            return this.ToString().GetHashCode();
        }
        public static bool operator ==(Point p1, Point
p2)
        {
            return p1.Equals(p2);
        }
        public static bool operator !=(Point p1, Point
p2)
        {
            return !(p1 == p2);
        }
        public static bool operator >(Point p1, Point
p2)
        {
            return Math.Sqrt(p1.X * p1.X + p1.Y * p1.Y)
>
            Math.Sqrt(p2.X * p2.X + p2.Y * p2.Y);
        }
        public static bool operator <(Point p1, Point
p2)
        {
            return Math.Sqrt(p1.X * p1.X + p1.Y * p1.Y)
<
            Math.Sqrt(p2.X * p2.X + p2.Y * p2.Y);
        }
```

Платформа Microsoft и C# (шарп) я программирования.

Перегрузка операторов отношений




```
public override string ToString()
        {
            return $"Point: X = {X}, Y = {Y}.";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Point point1 = new Point { X = 10, Y = 10 };
            Point point2 = new Point { X = 20, Y = 20 };
            WriteLine($"point1: {point1}");
            WriteLine($"point2: {point2}\n");
            WriteLine($"point1 == point2: { point1 ==
point2} "); // false
            WriteLine($"point1 != point2: {point1 !=
point2}\n"); // true
            WriteLine($"point1 > point2: {point1 >
point2}"); // false
            WriteLine($"point1 < point2: {point1 <
point2}"); // true
        }
    }
}
```

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов true и false

При перегрузке операторов true и false разработчик задает критерий истинности для своего типа данных. После этого, объекты типа напрямую можно использовать в структуре операторов if, do, while, for в качестве условных выражений.
Перегрузка выполняется по следующим правилам:

- **оператор true должен возвращать значение true, если состояние объекта истинно и false в противном случае;**
- **оператор false должен возвращать значение true, если состояние объекта ложно и false в противном случае;**
- **операторы true и false надо перегружать в паре.**

В качестве критерия истинности мы взяли равенство всех координат конкретной точки нулевому значению.

```
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
        public static bool operator true(Point p)
        {
            return p.X != 0 || p.Y != 0 ? true : false;
        }
        public static bool operator false(Point p)
        {
            return p.X == 0 && p.Y == 0 ? true : false;
        }
        public override string ToString()
        {
            return $"Point: X = {X}, Y = {Y}.";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            WriteLine("Введите координаты точки на
плоскости");
            Point point = new Point
            {
                X = int.Parse(ReadLine()),
                Y = int.Parse(ReadLine())
            };
            if (point)
            {
                WriteLine("Точка не в начале координат.");
            }
            else
            {
                WriteLine("Точка в начале координат.");
            }
        }
    }
}
```

21

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) языки программиров

Перегрузка логических операторов

Логические операторы && и || перегрузить нельзя, но они моделируются с помощью операторов & и |, допускающих перегрузку. Для того чтобы это стало возможным, необходимо выполнить ряд требований:

- **в классе должна быть выполнена перегрузка операторов true и false;**
- **в классе необходимо перегрузить логические операторы & и |;**
- **методы перегрузки операторов & и | должны возвращать тип класса, в котором осуществляется перегрузка;**
- **параметрами в методах перегрузки операторов & и | должны быть ссылки на класс, который содержит перегрузку.**

```
using static System.Console;
namespace SimpleProject
{
    class Point
    {
        public int X { get; set; }
        public int Y { get; set; }
        public static bool operator true(Point p)
        {
            return p.X != 0 || p.Y != 0 ? true : false;
        }
        public static bool operator false(Point p)
        {
            return p.X == 0 && p.Y == 0 ? true : false;
        }
        // перегружаем логический оператор |
        public static Point operator |(Point p1,
        Point p2)
        {
            if ((p1.X != 0 || p1.Y != 0) || (p2.X !=
            0 || p2.Y != 0))
                return p2;
            return new Point();
        }
        // перегружаем логический оператор &
        public static Point operator &(Point p1,
        Point p2)
        {
            if ((p1.X != 0 && p1.Y != 0) && (p2.X !=
            0 && p2.Y != 0))
                return p2;
            return new Point();
        }
        public override string ToString()
        {
            return $"Point: X = {X}, Y = {Y}.";
        }
    }
```

```
class Program
  {
      static void Main(string[] args)
      {
          Point point1 = new Point { X = 10, Y = 10 };
          Point point2 = new Point { X = 0, Y = 0 };
          WriteLine($"point1: {point1}");
          WriteLine($"point2: {point2}\n");
          Write("point1 && point2: ");
          if (point1 && point2)
          {
              WriteLine("true");
          }
          else
          {
              WriteLine("false");
          }
          Write("point1 || point2: ");
          if (point1 || point2)
          {
              WriteLine("true");
          }
          else
          {
              WriteLine("false");
          }
      }
  }
}
```

```
Консоль отладки Microsoft Visual Studio
point1: Point: X = 10, Y = 10.
point2: Point: X = 0, Y = 0.

point1 && point2: false
point1 || point2: true
```

https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/boolean-logical-operators

Платформа Microsoft и C# (шарп) язык программирования

Перегрузка логических операторов

- Унарный ! (логическое отрицание) оператор.
- Бинарные & (логическое И), | (логическое ИЛИ), а также ^ (логическое исключающее ИЛИ) операторы. Эти операторы всегда обрабатывают оба операнда.
- Бинарные && (условное логическое И) и || (условное логическое ИЛИ) операторы. Эти операторы вычисляют правый операнд, только если это необходимо.

Выполнение операторов && и || происходит следующим образом. Для оператора && первый операнд проверяется с использованием перегруженного оператора false, если результат равен false, тогда дальнейшее сравнение операндов осуществляется с использованием перегруженного оператора &, результат этого сравнения проверяется вызовом перегруженного оператора true, так как используется условный оператор. Если результат оператора false для первого операнда будет равен true, тогда оператор & выполняться не будет, а параметром для оператора true будет являться первый операнд. Для оператора || первый операнд проверяется с использованием перегруженного оператора true, если результат равен false, тогда дальнейшее сравнение операндов осуществляется с использованием перегруженного оператора |, результат этого сравнения также проверяется вызовом перегруженного оператора true (условный оператор). Если результат оператора true для первого операнда будет равен true, тогда оператор | выполняться не будет, а параметром для оператора true будет являться первый операнд.

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов преобразования

В собственных типах можно определить операторы, которые будут использоваться для выполнения приведения. Приведение может быть двух типов:

- **из произвольного типа в собственный тип;**
- **из собственного типа в произвольный тип.**

Для ссылочных и значимых типов приведение выполняется одинаково.

Как Вам известно, приведение может выполняться явным и неявным образом. Явное приведение типов требуется, если возможна потеря данных в результате приведения. Например:

- **при преобразовании int в short, потому что размер short недостаточен для сохранения значения int;**
- **при преобразовании типов данных со знаком в без знаковые может быть получен неверный результат, если переменная со знаком содержит отрицательное значение;**
- **при конвертировании типов с плавающей точкой в целые, так как дробная часть теряется;**
- **при конвертировании типа, допускающего null значения, в тип, не допускающий null, если исходная переменная содержит null, генерируется исключение.**

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов преобразования

Если потери данных в результате приведения не происходит, приведение можно выполнять как неявное. Операция приведения должна быть помечена либо как implicit, либо как explicit, чтобы указать, как ее предполагается использовать:

- **implicit задает неявное преобразование, его можно использовать, если преобразование всегда безопасно независимо от значения переменной, которая преобразуется;**
- **explicit задает явное преобразование, его следует использовать, если возможна потеря данных или возникновение исключения.**

Объявление оператора преобразования в классе:

```
public static {implicit|explicit}
                operator целевой_тип (исходный_тип)
{
    // код
}
```

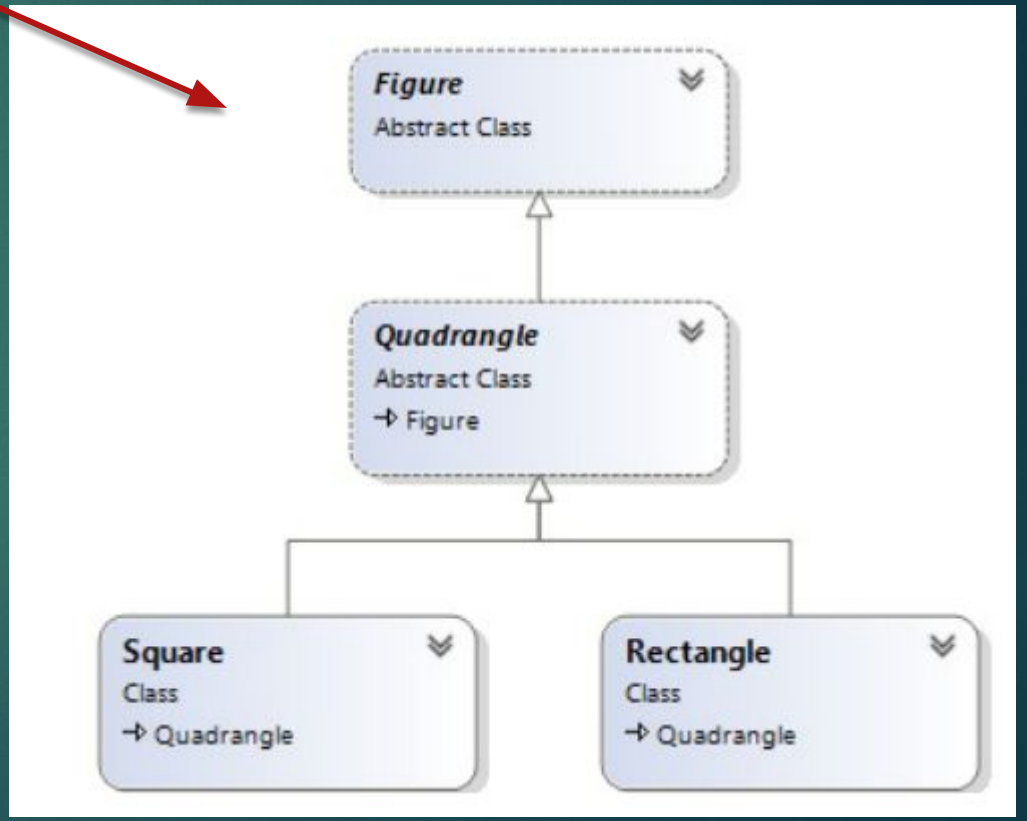
Имеется возможность выполнять приведение между экземплярами разных собственных структур или классов. Однако при этом существуют следующие ограничения:

- **нельзя определить приведение между классами, если один из них является наследником другого;**
- **приведение может быть определено только в одном из типов: либо в исходном типе, либо в типе назначения.**

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов преобразования

Например, имеется следующая иерархия классов. Единственное допустимое приведение типов — это приведения между классами Square и Rectangle, потому что эти классы не наследуют друг друга. При этом следует помнить, что если оператор преобразования определен внутри одного класса, то нельзя определить такой же оператор внутри другого класса. Приведем пример использования операторов преобразования, взяв за основу иерархию классов на рисунке. Перегрузим, оператор неявного преобразования (implicit), из класса Square в класс Rectangle без потери данных — ширину и высоту прямоугольника получаем на основании стороны квадрата. Также перегрузим, оператор явного преобразования (explicit), из класса Rectangle в класс Square с потерей данных — сторона квадрата равна высоте прямоугольника, ширина не учитывается. Для класса Square определим явное и неявное преобразование к целому типу.



Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов преобразования

```
using static System.Console;
namespace SimpleProject
{
    abstract class Figure
    {
        public abstract void Draw();
    }
    abstract class Quadrangle : Figure { }
    class Rectangle : Quadrangle
    {
        public int Width { get; set; }
        public int Height { get; set; }
        public static implicit operator Rectangle(Square s)
        {
            return new Rectangle
            {
                Width = s.Length * 2,
                Height = s.Length
            };
        }
        public override void Draw()
        {
            for (int i = 0; i < Height; i++, WriteLine())
            {
                for (int j = 0; j < Width; j++)
                {
                    Write("*");
                }
            }
            WriteLine();
        }
        public override string ToString()
        {
            return $"Rectangle: Width = {Width}, Height = { Height}";
        }
    }
```

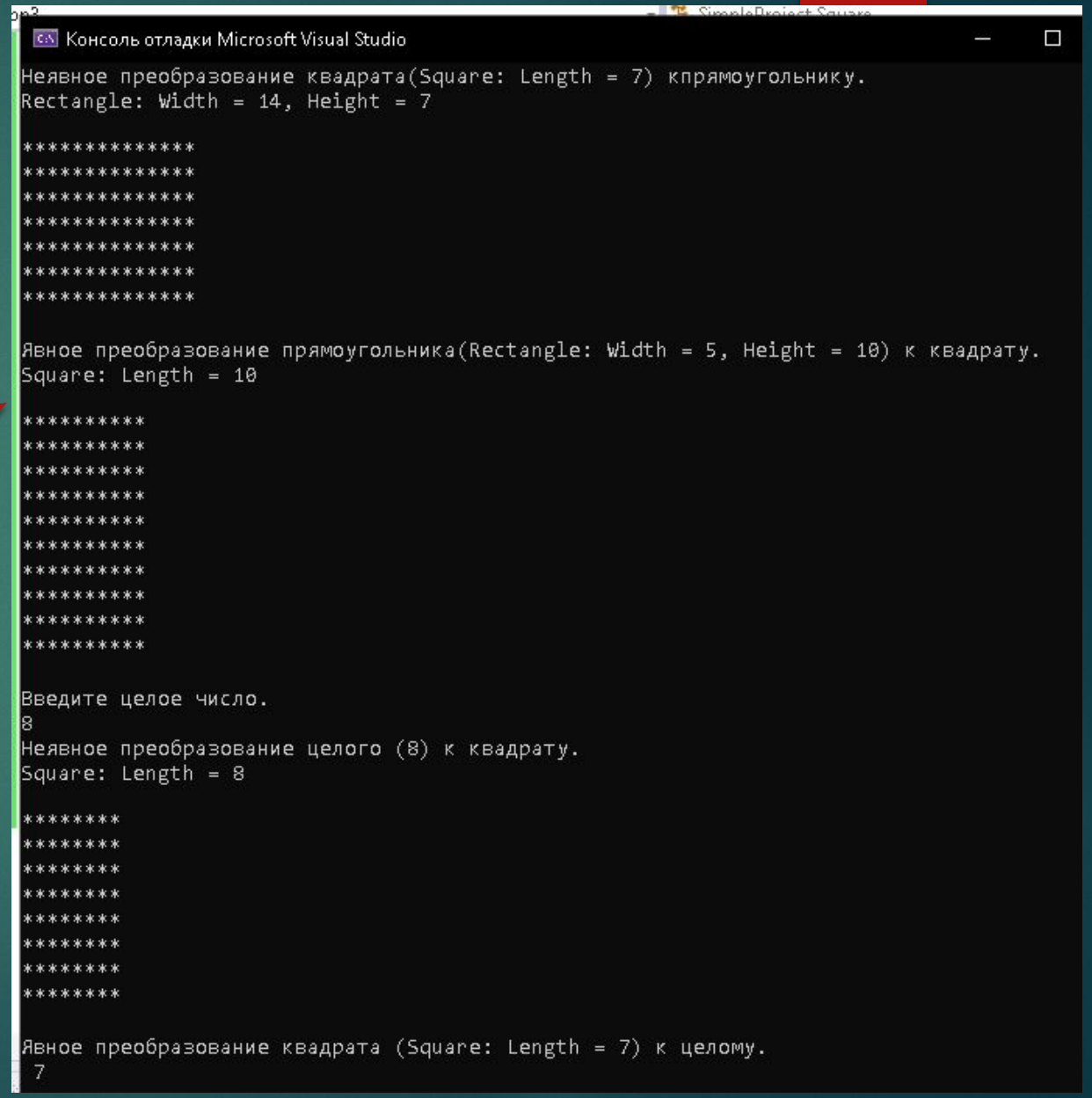
```
class Square : Quadrangle
    {
        public int Length { get; set; }
        public static explicit operator
        Square(Rectangle rect)
        {
            return new Square { Length = rect.Height };
        }
        public static explicit operator int(Square s)
        {
            return s.Length;
        }
        public static implicit operator Square(int number)
        {
            return new Square { Length = number };
        }
        public override void Draw()
        {
            for (int i = 0; i < Length; i++, WriteLine())
            {
                for (int j = 0; j < Length; j++)
                {
                    Write("*");
                }
            }
            WriteLine();
        }
        public override string ToString()
        {
            return $"Square: Length = {Length}";
        }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            Rectangle rectangle = new Rectangle
            {
                Width = 5,
                Height = 10
            };
            Square square = new Square { Length = 7 };
            Rectangle rectSquare = square;
            WriteLine($"Неявное преобразование квадрата
({ square}) кпрямоугольнику.\n{ rectSquare}\n");
            rectSquare.Draw();
            Square squareRect = (Square)rectangle;
            WriteLine($"Явное преобразование
прямоугольника({ rectangle}) к квадрату.\n{
squareRect}\n");
            squareRect.Draw();
            WriteLine("Введите целое число.");
            int number = int.Parse(ReadLine());
            Square squareInt = number;
            WriteLine($"Неявное преобразование целого ({
number}) к квадрату.\n{ squareInt}\n");
            squareInt.Draw();
            number = (int)square;
            WriteLine($"Явное преобразование квадрата ({
square}) к целому.\n { number} ");
        }
    }
}
```

Платформа Microsoft и C# (шарп) язык программирования.

Перегрузка операторов преобразования

Результат выполнения программы!



Платформа Microsoft и C# (шарп) язык программирования.

Индексаторы

А теперь ещё одно любопытное средство языка C#,представляющее собой одновременно способ перегрузки оператора [] (но без участия ключевого слова operator) и разновидность свойства (или как его ещё называют, свойством с параметрами). Индексаторы применяются для облегчения работы со специальными классами, реализующими пользовательскую коллекцию, используя синтаксис индексирования массива. Объявление индексатора подобно свойству, но с той разницей, что индексаторы безымянные (вместо имени используется ссылка this) и что индексаторы включают параметры индексирования.

Синтаксис объявления индексатора следующий:

```
тип_данных this[тип_аргумента] {get; set;}
```

Тип_данных — это тип объектов коллекции, this — это ссылка на объект, в котором определен индексатор. То, что для индексаторов используется синтаксис со ссылкой this, подчёркивает, что их можно использовать только на экземплярном уровне и никак иначе. Тип_аргумента представляет индекс объекта в коллекции, причём этот индекс необязательно целочисленный, он может быть любого типа. У каждого индексатора должен быть минимум один параметр, но их может быть и больше (многомерные индексаторы).

Платформа Microsoft и C# (шарп) язык программирования.

Индексаторы

Создание одномерных индексаторов

```
тип_данных this[тип_аргумента] {get; set;}
```

Рассмотрим пример создания и применения индексатора. Предположим, есть некий магазин (класс Shop), занимающийся реализацией ноутбуков (класс Laptop). Дабы не перегружать пример лишней информацией, снабдим класс Laptop только двумя свойствами: Vendor — имя фирмы-производителя и Price — цена ноутбука. Также переопределим метод ToString() для отображения информации по конкретной единице товара. в качестве единственного поля класса Shop выступает ссылка на массив объектов Laptop. в конструкторе с одним параметром задаётся количество элементов массива и выделяется память для их хранения. Далее нам нужно сделать возможным обращение к элементам этого массива через экземпляр класса Shop, пользуясь синтаксисом массива так, словно класс Shop и есть массив элементов типа Laptop. Для этого мы добавляем в класс Shop индексатор.

```
1  public Laptop this[int index]
2  {
3      get
4      {
5          if (index >= 0 && index < laptopArr.Length)
6          {
7              return laptopArr[index];
8          }
9          throw new IndexOutOfRangeException();
10     }
11     set
12     {
13         laptopArr[index] = value;
14     }
15 }
```

Платформа Microsoft и C# (шарп) язык программирования.

Индексаторы

Создание одномерных индексаторов

```
тип_данных this[тип_аргумента] {get; set;}
```

Здесь в аксессоре get осуществляется проверка нахождения индекса в пределах массива и в случае если в индексатор попробуют передать индекс, который находится за пределами массива, тогда будет сгенерирована исключительная ситуация IndexOutOfRangeException. Еще одна особенность данной программы — свойство Length в классе Shop, добавление которого позволяет получать размер массива laptopArr класса Shop подобно свойству Length стандартного массива.

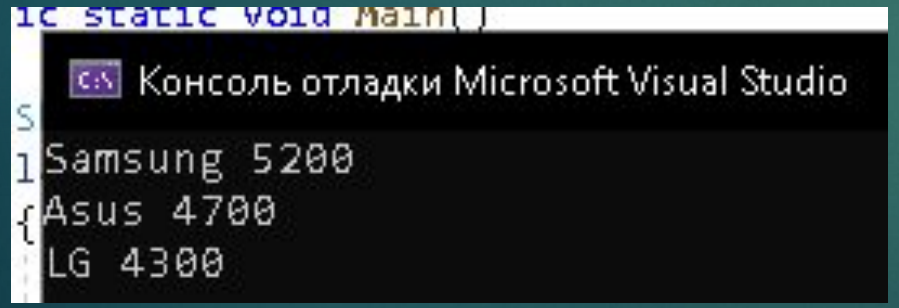
```
1   public int Length
2   {
3   get { return laptopArr.Length; }
4   }
```

Платформа Microsoft и C# (шарп) язык программирования.

Индексаторы

Создание одномерных индексат

Теперь рассмотрим код программы целиком.




```
using System;
using static System.Console;
namespace SimpleProject
{
    public class Laptop
    {
        public string Vendor { get; set; }
        public double Price { get; set; }
        public override string ToString()
        {
            return $"{Vendor} {Price}";
        }
    }
    public class Shop
    {
        Laptop[] laptopArr;
        public Shop(int size)
        {
            laptopArr = new Laptop[size];
        }
        public int Length
        {
            get { return laptopArr.Length; }
        }
        public Laptop this[int index]
        {
            get
            {
                if (index >= 0 && index <
                laptopArr.Length)
                {
                    return laptopArr[index];
                }
                throw new IndexOutOfRangeException();
            }
            set
            {
                laptopArr[index] = value;
            }
        }
    }
```

```
public class Program
  {
      public static void Main()
      {
          Shop laptops = new Shop(3);
          laptops[0] = new Laptop
          {
              Vendor = "Samsung",
              Price = 5200
          };
          laptops[1] = new Laptop
          {
              Vendor = "Asus",
              Price = 4700
          };
          laptops[2] = new Laptop
          {
              Vendor = "LG",
              Price = 4300
          };
          try
          {
              for (int i = 0; i < laptops.Length; i++)
              {
                  WriteLine(laptops[i]);
              }
          }
          catch (Exception ex)
          {
              WriteLine(ex.Message);
          }
      }
  }
}
```

Платформа Microsoft и C# (шарп) язык программирования.

ШАГ компьютерная АКАДЕМИЯ

Индексаторы

Создание многомерных индексаторов

В C# есть возможность создавать не только одномерные, но и многомерные индексаторы. Это возможно, если класс-контейнер содержит в качестве поля массив с более чем одним измерением. Для демонстрации такой возможности приведём схематичный пример использования двумерного индексатора, не перегруженный дополнительными проверками.

```
using static System.Console;
namespace SimpleProject
{
    public class MultArray
    {
        private int[,] array;
        public int Rows { get; private set; }
        public int Cols { get; private set; }
        public MultArray(int rows, int cols)
        {
            Rows = rows;
            Cols = cols;
            array = new int[rows, cols];
        }
        public int this[int r, int c]
        {
            get { return array[r, c]; }
            set { array[r, c] = value; }
        }
    }
```

```
public class Program
    {
        static void Main()
        {
            MultArray multArray = new
MultArray(2, 3);
            for (int i = 0; i < multArray.Rows;
i++)
            {
                for (int j = 0; j < multArray.Cols;
j++)
                {
                    multArray[i, j] = i + j;
                    Write($"{multArray[i, j]} ");
                }
                WriteLine();
            }
        }
    }
}
```

Платформа Microsoft и C# (шарп) язык программирования.

Индексаторы

Перегрузка индексаторов

Как было отмечено ранее, тип может поддерживать различные перегрузки индексаторов при условии, что они отличаются сигнатурой. Это значит, что тип параметра индексатора может быть не только целочисленным, но и вообще любым. Например, можно добавить в наш класс индексатор для поиска по имени производителя. Для этого мы создали перечисление Vendors, при вводе производителя поиск совпадения будет осуществляться среди значений этого перечисления. Также мы добавили в наш класс индексатор для поиска по цене. Для поиска индекса элемента по заданной цене был создан дополнительный метод FindByPrice(). в аксессорах set наш код никак не реагирует на неправильный ввод значений, он просто игнорируется. Теперь у нас есть три перегрузки индексатора, но они между собой не конфликтуют, поскольку их сигнатуры не совпадают по типу данных.

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

И

```
using System;
using static System.Console;
namespace SimpleProject
{
    public class Laptop
    {
        public string Vendor { get; set; }
        public double Price { get; set; }
        public override string ToString()
        {
            return $"{Vendor} {Price}";
        }
    }
    enum Vendors { Samsung, Asus, LG };
    public class Shop
    {
        private Laptop[] laptopArr;
        public Shop(int size)
        {
            laptopArr = new Laptop[size];
        }
        public int Length
        {
            get { return laptopArr.Length; }
        }
        public Laptop this[int index]
        {
            get
            {
                if (index >= 0 && index <
                laptopArr.Length)
                {
                    return laptopArr[index];
                }
                throw new IndexOutOfRangeException();
            }
            set
            {
                laptopArr[index] = value;
            }
        }
```

```
public Laptop this[string name]
    {
        get
        {
            if (Enum.IsDefined(typeof(Vendors), name))
            {
                return laptopArr[(int)Enum.
                Parse(typeof(Vendors), name)];
            }
            else
            {
                return new Laptop();
            }
        }
        set
        {
            if (Enum.IsDefined(typeof(Vendors), name))
            {
                laptopArr[(int)Enum.
                Parse(typeof(Vendors), name)] =
                value;
            }
        }
    }
    public int FindByPrice(double price)
    {
        for (int i = 0; i < laptopArr.Length; i++)
        {
            if (laptopArr[i].Price == price)
            {
                return i;
            }
        }
        return -1;
    }
    public Laptop this[double price]
    {
        get
        {
            if (FindByPrice(price) >= 0)
            {
                return this[FindByPrice(price)];
            }
            throw new Exception("Недопустимая стоимость.");
        }
        set
        {
            if (FindByPrice(price) >= 0)
            {
                this[FindByPrice(price)] = value;
            }
        }
    }
}
```

```
public class Program
  {
    public static void Main()
    {
        Shop laptops = new Shop(3);
        laptops[0] = new Laptop
        {
            Vendor = "Samsung",
            Price = 5200
        };
        laptops[1] = new Laptop
        {
            Vendor = "Asus",
            Price = 4700
        };
        laptops[2] = new Laptop
        {
            Vendor = "LG",
            Price = 4300
        };
        try
        {
            for (int i = 0; i < laptops.Length; i++)
            {
                WriteLine(laptops[i]);
            }
            WriteLine();
            WriteLine($"Производитель Asus: { laptops["Asus"]}.");
            WriteLine($"Производитель HP: { laptops["HP"]}.");
            // игнорирование
            laptops["HP"] = new Laptop();
            WriteLine($"Стоимость 4300: { laptops[4300.0]}.");
            // недопустимая стоимость
            WriteLine($"Стоимость 10500: { laptops[10500.0]}.");
            // игнорирование
            laptops[10500.0] = new Laptop();
        }
        catch (Exception ex)
        {
            WriteLine(ex.Message);
        }
    }
}
```

ШАГ компьютерная АКАДЕМИЯ

Платформа программир

Домашняя работа

Урок №5

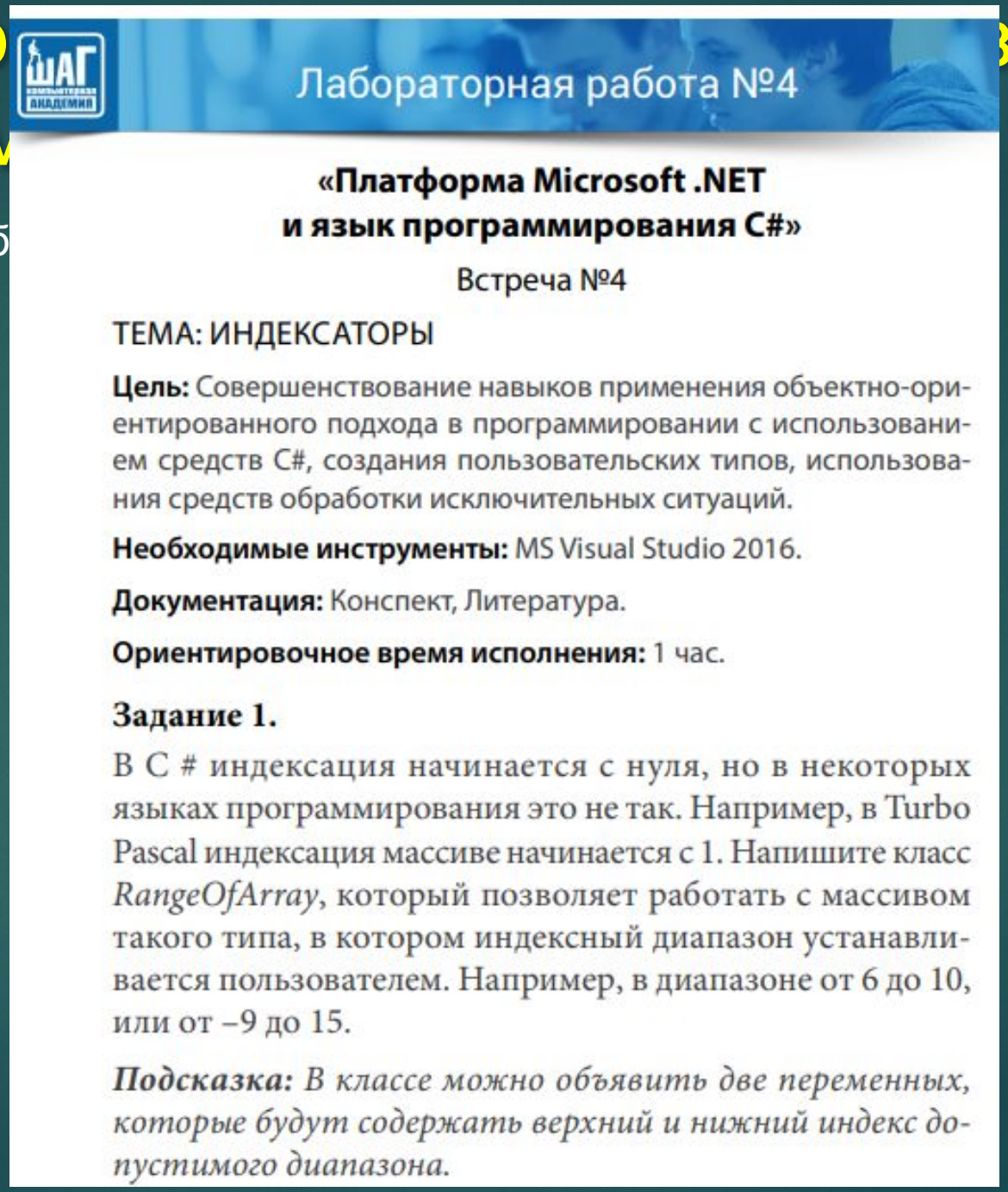
Домашнее задание

Разработать класс `Fraction`, представляющий простую дробь. в классе предусмотреть два поля: числитель и знаменатель дроби. Выполнить перегрузку следующих операторов: +,-,*,/,==,!=,<,>, `true` и `false`.
Арифметические действия и сравнение выполняется в соответствии с правилами работы с дробями. Оператор `true` возвращает `true` если дробь правильная (числитель меньше знаменателя), оператор `false` возвращает `true` если дробь неправильная (числитель больше знаменателя).
Выполнить перегрузку операторов, необходимых для успешной компиляции следующего фрагмента кода:

```
Fraction f = new Fraction(3, 4);
int a = 10;
Fraction f1 = f * a;
Fraction f2 = a * f;
double d = 1.5;
Fraction f3 = f + d;
```

Платфор… …зык программ…

Лабораторная раб…



Лабораторная работа №4

«Платформа Microsoft .NET и язык программирования C#»

Встреча №4

ТЕМА: ИНДЕКСАТОРЫ

Цель: Совершенствование навыков применения объектно-ориентированного подхода в программировании с использованием средств C#, создания пользовательских типов, использования средств обработки исключительных ситуаций.

Необходимые инструменты: MS Visual Studio 2016.

Документация: Конспект, Литература.

Ориентировочное время исполнения: 1 час.

Задание 1.

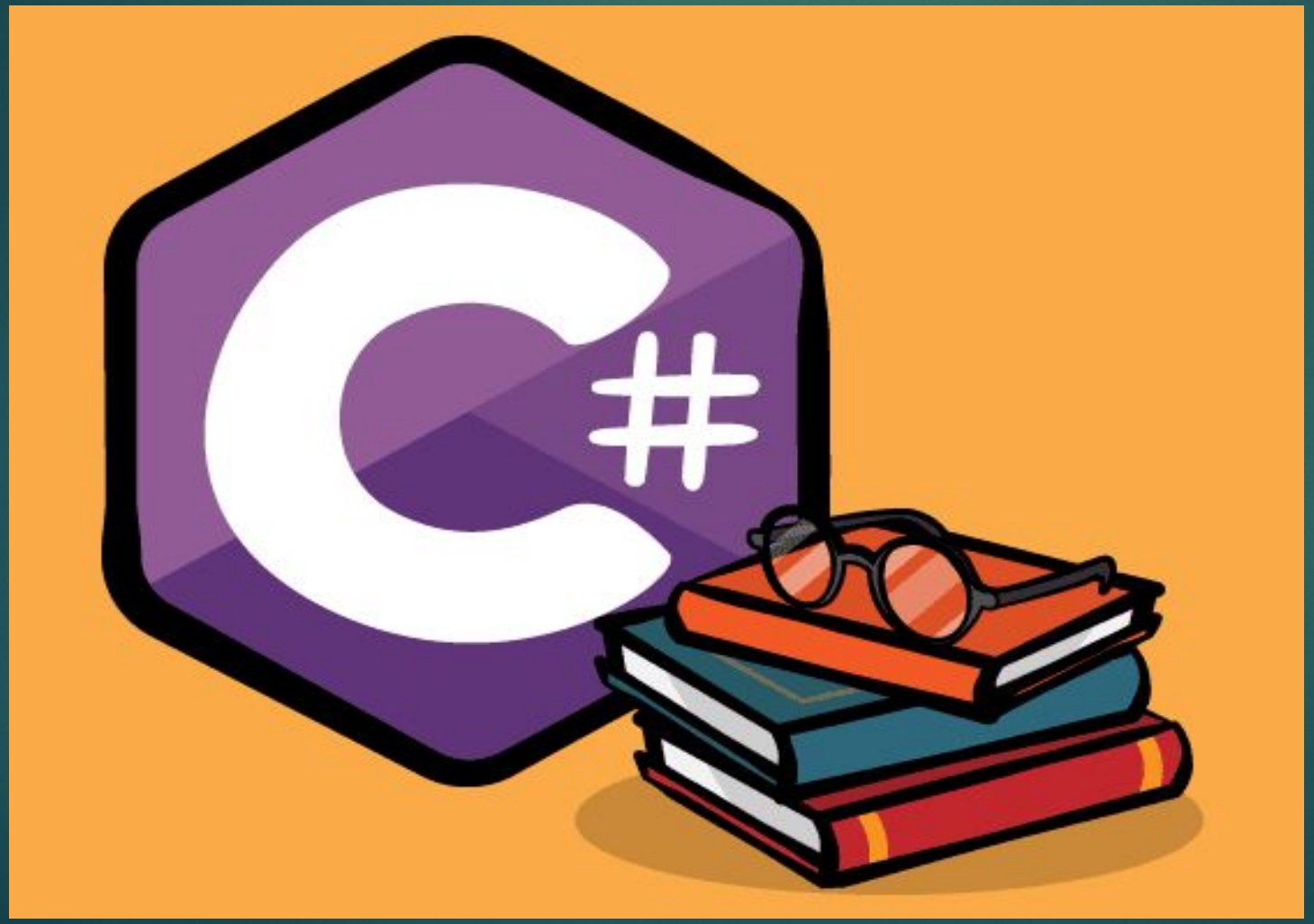
В C # индексация начинается с нуля, но в некоторых языках программирования это не так. Например, в Turbo Pascal индексация массиве начинается с 1. Напишите класс *RangeOfArray*, который позволяет работать с массивом такого типа, в котором индексный диапазон устанавливается пользователем. Например, в диапазоне от 6 до 10, или от –9 до 15.

Подсказка: *В классе можно объявить две переменных, которые будут содержать верхний и нижний индекс допустимого диапазона.*

ШАГ компьютерная АКАДЕМИЯ

Платформа Microsoft и C# (шарп) язык программирования.

Повторение предыдущих тем!



Платформа Microsoft и C# (шарп) язык программирования.

Модификаторы доступа в C#

В C# применяются следующие модификаторы доступа:

public: *публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.*

private: *закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.*

protected: *такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.*

internal: *класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).*

protected internal: *совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.*

private protected: *такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.*

Платформа Microsoft и C# (шарп) язык программирования.

Абстрактные классы и члены классов

Кроме обычных свойств и методов абстрактный класс может иметь абстрактные члены классов, которые определяются с помощью ключевого слова abstract и не имеют никакого функционала. В частности, абстрактными могут быть: Методы, Свойства, Индексаторы, События.

Абстрактные члены классов не должны иметь модификатор private. При этом производный класс обязан переопределить и реализовать все абстрактные методы и свойства, которые имеются в базовом абстрактном классе. При переопределении в производном классе такой метод или свойство также объявляются с модификатором override (как и при обычном переопределении виртуальных методов и свойств). Также следует учесть, что если класс имеет хотя бы одиный абстрактный метод (или абстрактные свойство, индексатор, событие), то этот класс должен быть определен как абстрактный.

Платформа Microsoft и C# (шарп) язык программирования.

Статические члены и модификатор static

Кроме обычных полей, методов, свойств класс может иметь статические поля, методы, свойства. Статические поля, методы, свойства относятся ко всему классу и для обращения к подобным членам класса необязательно создавать экземпляр класса.

На уровне памяти для статических полей будет создаваться участок в памяти, который будет общим для всех объектов класса.

Платформа Microsoft и C# (шарп) язык программирования.

Перечисления enum

В C# есть такой тип как **enum** или перечисление. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора **enum**. Далее идет название перечисления, после которого указывается тип перечисления - он обязательно должен представлять целочисленный тип (byte, int, short, long). Если тип явным образом не указан, то по умолчанию используется тип int. Затем идет список элементов перечисления через запятую:

```
1  enum Days
2  {
3      Monday,
4      Tuesday,
5      Wednesday,
6      Thursday,
7      Friday,
8      Saturday,
9      Sunday
10 }
11
12 enum Time : byte
13 {
14     Morning,
15     Afternoon,
16     Evening,
17     Night
18 }
```

0 1 2 3 4 5 6

Но можно и для всех элементов явным образом указать значения:

```
1  enum Operation
2  {
3      Add = 2,
4      Subtract = 4,
5      Multiply = 8,
6      Divide = 16
7  }
```

Также стоит отметить, что перечисление необязательно определять внутри класса, можно и вне класса, но в пределах пространства имен:

```
1  enum Operation
2  {
3      Add = 1,
4      Subtract,
5      Multiply,
6      Divide
7  }
8  class Program
9  {
10     static void Main(string[] args)
11     {
12         Operation op;
13         op = Operation.Add;
14         Console.WriteLine(op); // Add
15
16         Console.ReadLine();
17     }
18 }
```