Вопросы к зачету по дисциплинам:

«Объектно-ориентированные технологии программирования и стандарты проектирования»

«Объектно-ориентированное проектирование и программирование»

1. **Состав. NET Framework. Структура среды выполнения CLR.**

1.Net Framework – среда разработки и выполнения(платформа), которая создана Microsoft для разработки приложений

общеязыковая исполняющая среда (CLR) Common Language Runtime) Виртуальная Машина. Обеспечивает выполнение сборки (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация В данном случае под виртуальной машиной (Virtual Machine) понимается программная среда, которая эмулирует работу физического компьютера.

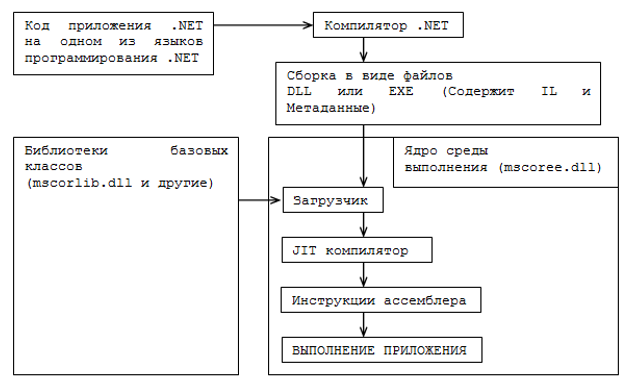
библиотека классов (FCL). (.NET Framework Class Library) Это объектно-ориентированная библиотека классов, интерфейсов и системы типов, соответствующая спецификации Common Language Specification (CLS). FCL включает в себя базовые классы (BCL) и дополнительные библиотеки, такие как Windows Forms, ASP.NET, ADO.NET и другие. System System.Data System.IO System.Text

Спецификация Common Language Specification (CLS) — это набор правил и соглашений, которые определяют, как должны быть реализованы классы, интерфейсы и типы в .NET Framework, чтобы они могли взаимодействовать друг с другом независимо от языка программирования.которые должен обязательно поддерживать каждый отдельно взятый .NET-компилятор

Структура среды выполнения CLR.

CTS (Common Type Systems)- спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Описывает все, что касается определения и поведения полей, методов, свойств, событий и т.д. CLS определяет правила и соглашения для обеспечения совместимости кода, а CTS определяет типы данных и операции с ними, обеспечивая единообразие типов во всех языках .NET.

CTS->CLS



JIT-компилятор (Just-In-Time Compiler) называется так потому, что он компилирует промежуточный язык (Intermediate Language, IL) в машинный код непосредственно перед выполнением программы. Это означает, что компиляция происходит "на лету" (just in time), когда код действительно нужен для выполнения.

1. **Структура управляемого модуля - portable executable (PE). Понятие и исполнение сборки. CIL.**

Структура равляемого модуля - portable executable (PE)

|  |
| --- |
| заголовок PE - информация для загрузки файла в память, |
| заголовок CLR (версия CLR, точки входа модуля, размеры и месторасположение ресурсов и метаданных), |
| метаданные (специальные таблицы, содержащие исходный код типов и членов данных); |
| код IL (код который CLR компилирует в команды процессора). |

Типы сборок

с нестрогими именами (weakly named assemblies)

со строгими именами (strongly named assemblies

Сборка (assembly) — 1) понятие, для группировки одного или нескольких управляемых модулей или файлов ресурсов. дискретная единица многократно используемого кода внутри CLR

Исполнение сборки

CLR ищет типы данных и загружает во внутренние структуры

Для каждого метода CLR заносит адрес внутренней CLR функции JITCompiler

JITCompiler ищет в метаданных соответствующей сборки IL-код вызываемого метода, проверяет и компилирует IL-код в машинные команды

Они хранятся в динамически выделенном блоке памяти.

JITCompiler заменяет адрес вызываемого метода адресом блока памяти, содержащего готовые машинные команды

JITCompiler передает управление коду в этом блоке памяти.

При компиляции кода компиляторы NET Framework генерируют код на промежуточном языке (CIL, Common Intermediate Language). При исполнении CLR транслирует CIL-код в команды соответствующего процессора.

1. **CTS (Common Type System). Типы данных C#. Ссылочные и типы значений.**

CTS (Common Type Systems)- спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Описывает все, что касается определения и поведения полей, методов, свойств, событий и т.д.

Типы данных в C#: В C# все типы данных делятся на две категории: ссылочные типы и типы значений.

Ссылочный тип: Object является ссылочным типом, что означает, что переменные типа Object содержат ссылки на данные, а не сами данные.

Ссылочные типы: - в куче, обрабат сборщиком мусора

Классы (Classes): Определяют объекты, которые могут содержать данные и методы. Пример: class Person { public string Name; }

Интерфейсы (Interfaces): Определяют контракты, которые классы должны реализовать. Пример: interface IAnimal { void MakeSound(); }

Делегаты (Delegates): Представляют методы и позволяют передавать их как параметры. Пример: delegate void MyDelegate(string message);

Массивы (Arrays): Хранят коллекции элементов одного типа. Пример: int[] numbers = new int[5];

Строки (Strings): Представляют последовательности символов. Пример: string greeting = "Hello";

Типы значений:- в стеке, не обрабат сбошиком мусора

Простые типы (Simple Types): Включают в себя числовые типы (int, float, double), логический тип (bool) и символы (char). Пример: int age = 30;

Структуры (Structs): Определяют типы данных, которые содержат значения. Пример: struct Point { public int X; public int Y; }

Перечисления (Enums): Определяют набор именованных констант. Пример: enum Days { Sunday, Monday, Tuesday }



1. **Понятие упаковки и распаковки типов. Типы Nullable: преобразование, проверка, null-объединение**

Упаковкой (boxing) называется процесс преобразования типа значения в тип System.Object или в тип интерфейса, который реализуется данным типом-значением

в управляемой куче выделяется память

поля копируются

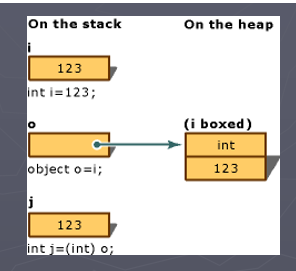
возвращается адрес объекта

Объекты значимого типа существуют в двух формах: неупакованной (unboxed) и упакованной (boxed). Ссылочные типы бывают только в упакованной форме.

распаковка (unboxing)

получение указателя на исходный значимый тип (поля данных), содержащийся в объекте

\*1. В управляемой куче выделяется память. Ее объем определяется длиной значимого типа и двумя дополнительными членами — указателем на типовой объект и индексом блока синхронизации. Эти члены необходимы для всех объектов в управляемой куче.



Цель это использовние типов значений в коллекуиях и вызов методов для Object.

**Типы Nullable**:

Типы Nullable позволяют значениям типов значений (например, int, float) принимать значение null.

Преобразование 2 способа

**int? x = null;**

**Nullable<int> x = 5; -только для типов знач**

Проверка

Value - значение объекта

HasValue: возвращает true, если объект хранит некоторое значение, и false, если объект равен null.

**null-объединение**

применяется для установки значений по умолчанию для типов значений и ссылочных типов, которые допускают значение null

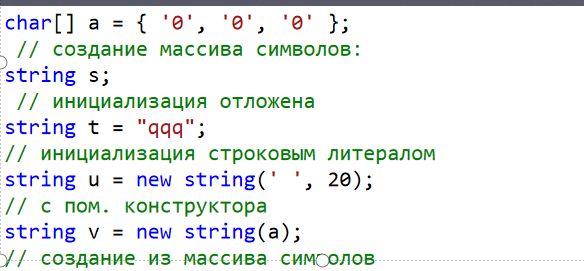
**int? z = 2;**

**int t = z ?? 1;---**возвращает левый операнд, если этот операнд не равен null

Иначе возвращается правый операнд

1. **Тип данных String: операции, литералы, пустые и нулевые строки, форматированный вывод.**

Тип string предназначен для работы со строками символов в кодировке Unicode. Ему соответствует базовый класс System.String библиотеки .NET



присваивание (=);

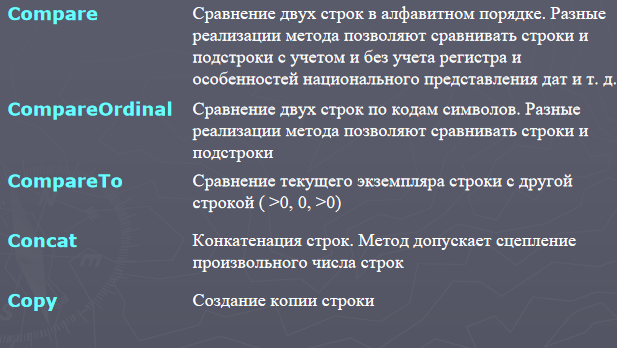
проверка на равенство содержимого (==);

проверка на неравенство (!=);

обращение по индексу ([]);

сцепление (конкатенация) строк (+)

<,>, >=,<= - сравнивают ссылки!!!!!!!! text.Replace("World", "C#");text.IndexOf("World"); text.Substring(0, 5);

\*\*\*\*\*

Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.

Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения.

строки типа string относятся к неизменяемым типам данных.

Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

**Литералы строк — это текстовые значения, заключенные в двойные кавычки.**

В случае, если литерал есть путь к файлу (локальный или сетевой), каталогу или web-странице, можно использовать альтернативный символ '/' вместо “\\”.

**пустые и нулевые строки**

Пустая строка — экземпляр объекта System.String, содержащий 0 символов:

string s = "";

Для пустых строк можно вызывать методы.

Строки со значениями null не ссылаются на экземпляр объекта System.String, попытка вызвать метод для строки null вызовет исключение NullReferenceException.

строки null можно использовать в операциях объединения и сравнения с другими строками.

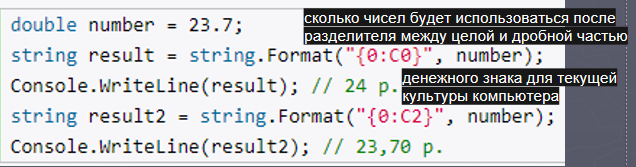
**форматированный вывод.**

Форматированный вывод позволяет вставлять значения переменных в строку.

string message = string.Format("Name: {0}, Age: {1}", name, age);

string message = $"Name: {name}, Age: {age}";

\*\*{index[,alignment][:formatString]}



В отличие от класса String, который создает новый объект при каждом изменении строки, StringBuilder изменяет существующий объект, что значительно улучшает производительность.

1. **Неявная типизация – назначение и использование.**

Неявная типизация в C# позволяет компилятору автоматически определять тип переменной на основе присваиваемого ей значения. Это делается с помощью ключевого слова var.

· **Упрощение кода**: Сокращает количество кода, который нужно писать, особенно при работе с длинными или сложными типами.

· **Улучшение читаемости**: Делает код более понятным, так как тип переменной очевиден из контекста.

· **Удобство**: Упрощает работу с анонимными типами и LINQ-запросами

### Важные моменты:

Тип переменной определяется на этапе компиляции и не может быть изменен.

Неявная типизация не означает динамическую типизацию. Тип переменной остается статически определенным.

var можно использовать только для локальных переменных, но не для полей класса или параметров метода.

var number = 10; // Компилятор определяет тип как int

var text = "Hello, world!"; // Компилятор определяет тип как string

var list = new List<string>(); // Компилятор определяет тип как List<string>

1. **Массивы C# одномерные, прямоугольные и ступенчатые.**

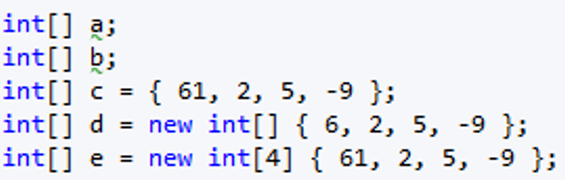
System.Array и System.Object

Ссылочный тип - в куче

int []w = new int[10];

int sizeW= w.Length

**Одномерные**

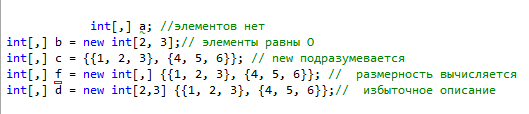


Console. WriteLine( numbers [3] ) ;

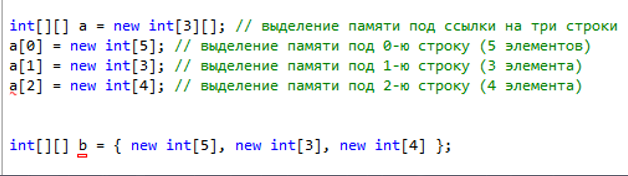
for(int i= 0; < numbers.Length; i++) ;

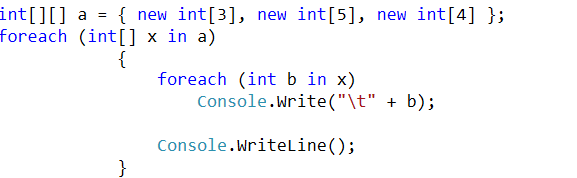
foreach ( int х in е ) Console.WriteLine

**Прямоугольные**



**ступенчатые.**





//принципиально знание характеристик первой составляю•щей!

1. **Понятие кортежей. Свойства, создание**

Кортежи (tuples) в C# — это структуры данных, которые позволяют хранить наборы значений различных типов. Они полезны, когда нужно вернуть несколько значений из метода или сгруппировать несколько значений вместе без создания отдельного класса или структуры

.

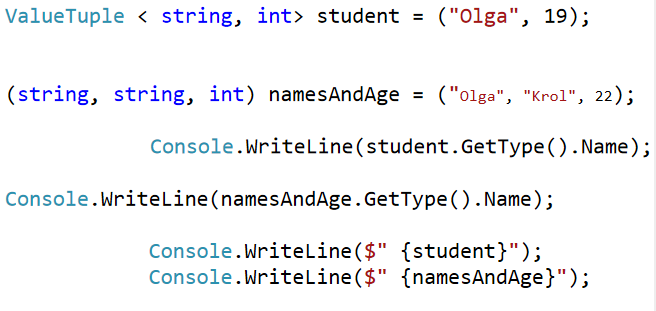
Свойства кортежей:

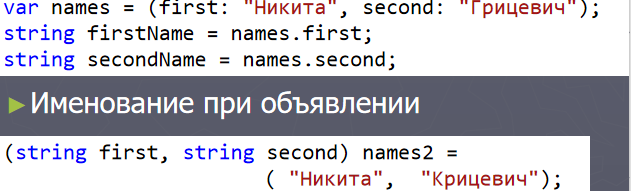
создается один раз и остается неименным (все свойства доступны только для чтения)

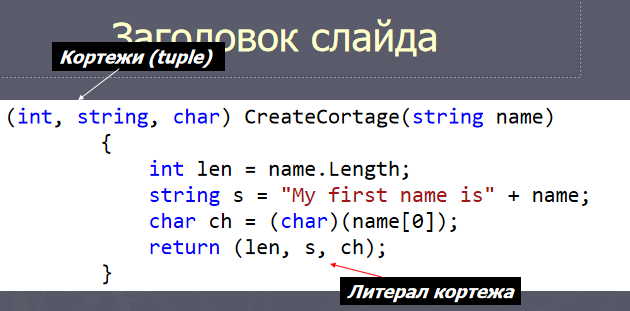
позволяют использовать методы CompareTo, Equals, GetHashCode и ToString, свойство Size

реализуют интерфейсы IStructuralEquatable, IStructuralComparable и IComparable (можно сравнивать)

КОРТЕДЖИ ЭТО ЗНАЧИМЫЙ ТИП







// Упаковка кортежа var tuple = (1, "Hello", true);

// Распаковка кортежа (int number, string text, bool flag)

1. **Принципы объектно-ориентированного программирования.**

Ооп в целом это это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования

**Инкапсуляция (Encapsulation);**

механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования

Совместное хранение данных и функций

Сокрытие внутренней информации от пользователя

Изоляция пользователя от особенностей реализаци

**Наследование (Inheritance);**

процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта.

иерархии классов

**Полиморфизм (Polymorphism);**

-поволяет методам классов иметь не одну, а несколько форм, и он необходим, когда у нас есть много классов, связанных друг с другом путем наследования

- способность программы идентично использовать объекты с одинаковым интерфейсом без информации о конкретном типе этого объекта

Поддержка полиморфизма осуществляется через виртуальные функции, механизм перегрузки функций и операторов, а также обобщения

**Абстракция данных(Abstraction).**

Абстракция подразумевает разделение и независимое рассмотрение интерфейса и реализации

способ выделить набор наиболее важных атрибутов и методов и исключить незначимые.

1. **Класс. Элементы класса. Свойства и индексаторы.**

Класс — это шаблон или чертеж для создания объектов. Он определяет свойства и методы, которые будут у объектов этого класса. Класс может содержать поля, методы, свойства, конструкторы, события и другие члены.

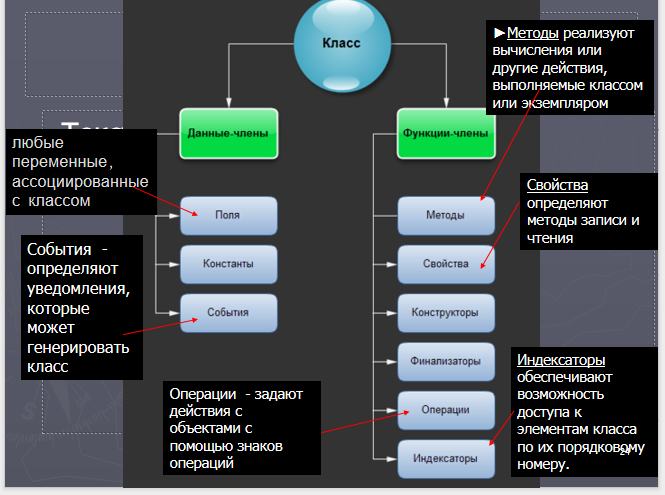
класс представляет новый тип, который определяется пользователем

[ атрибуты ] [ спецификаторы ]

class имякласса [ : предок ]

{

тело-класса }



**Свойства**

Свойства позволяют безопасно читать и изменять значения полей класса. Они состоят из методов доступа get и set.

public class Person

{

private string name;

private int age;

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

}

1) не может быть передано методу в качестве параметра ref или out.

2) не подлежит перегрузке

### Индексаторы

Индексаторы позволяют объектам класса работать как массивы, предоставляя доступ к элементам по индексу.

public class Person

{

// Массив имен

private string[] names = new string[5];

// Индексатор для доступа к именам

public string this[int index]

{

get { return names[index]; } // Получение имени по индексу

set { names[index] = value; } // Установка имени по индексу

}

}

// Создание экземпляра класса Person

Person person = new Person();

// Установка имен с помощью индексатора

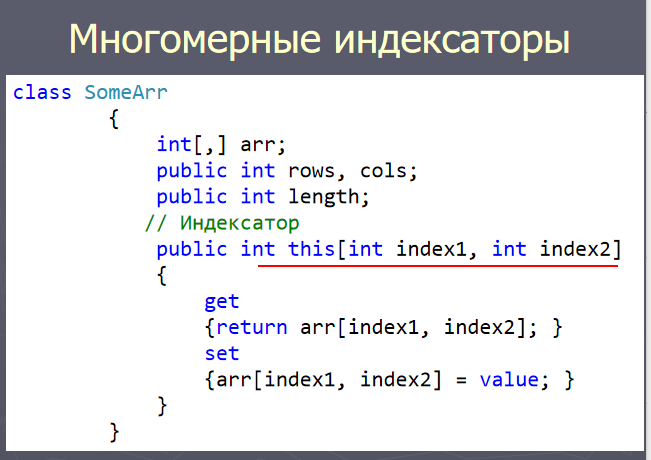
person[0] = "Alice";

person[1] = "Bob"; person[2] = "Charlie";

Ограничения на индексаторы:

значение, выдаваемое индексатором, нельзя передавать методу в качестве параметра ref или out

индексатор не может быть объявлен как static



1. **Класс. Константы. Поля только для чтения. Инициализаторы класса.**

### Константы

Константы — это значения, которые не могут быть изменены после их инициализации. Они объявляются с использованием ключевого слова const и должны быть инициализированы при объявлении.

public const int MyConstant = 10;

### Поля только для чтения

Поля только для чтения (readonly) — это поля, значения которых можно задать только при их объявлении или в конструкторе. После этого их значения не могут быть изменены.

public class MyClass

{

public readonly int MyReadOnlyField

public MyClass(int value)

{

MyReadOnlyField = value;

}

}



ИНИЦИАЛИЗАТОРЫ

С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.

class Student

{

public string name ;

public int age;

}

static void Main(string[] args)

{

// используем инициализаторы

Student someStud =

new Student {name = "Kate", age = 100};

}

С помощью инициализатора мы можем установить значения только доступных извне класса полей и свойств объекта.

Инициализатор выполняется после конструктора, поэтому если и в конструкторе, и в инициализаторе устанавливаются значения одних и тех же полей и свойств, то значения, устанавливаемые в конструкторе, заменяются значениями из инициализатора.

1. **Спецификаторы доступа C#. Видимость типов. Доступ к членам типов.**



Видимость типа может быть открытым (public) или внутренним (internal).- по умолч

// Внутренний тип доступен только из собственной сборки

// Открытый тип доступен из любой сборки

\*\*Контекст класса. Переменные, определенные на уровне класса, доступны в любом методе этого класса. Их еще называют глобальными переменными или полями

Контекст метода. Переменные, определенные на уровне метода, являются локальными и доступны только в рамках данного метода. В других методах они недоступны

Контекст блока кода. Переменные, определенные на уровне блока кода, также являются локальными и доступны только в рамках данного блока. Вне своего блока кода они не доступны.

При работе с переменными надо учитывать, что локальные переменные, определенные в методе или в блоке кода, скрывают переменные уровня класса, если их имена совпадают:

1. **Класс. Конструкторы и их свойства. Деструкторы**

Конструкторы — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа.

Память всегда обнуляется до вызова конструктора экземпляра типа. Любые поля, не задаваемые конструктором явно, гарантированно содержат 0 или null.

public Student() {

name = "IR234";

secondName = "Intel";

course = 1;

}

СВОЙСТВА

1) имя такое же как и имя типа (класса )

2) не имеет возвращаемого значения

3) не наследуются

4) нельзя применять модификаторы virtual, new, override, sealed и abstract

5) для класса без явно заданных конструкторов компилятор создает конструктор по умолчанию (без параметров)

6) для статических классов компилятор не создает конструктор по умолчанию

**7) может определяться несколько конструкторов, сигнатуры и уровни доступа к конструкторам обязательно должны отличаться**

8) можно явно заставлять один конструктор вызывать другой конструктор посредством зарезервированного слова this: (this обеспечивает доступ к текущему экземпляру класса

в любой нестатический метод автоматически передается скрытый параметр this)

public Student(int iCourse):this()

{ }

Отсутствие параметров: Конструктор по умолчанию не принимает никаких параметров.

**Деструкторы**

вызываться непосредственно перед окончательным уничтожением объекта системой "сборки мусора", чтобы гарантировать четкое окончание срока действия объекта.

нельзя узнать, когда именно вызовется деструктор

Если программа завершиться до того, как произойдет "сборка мусора", деструктор может быть вообще не вызван

~Student()

{

Console.WriteLine("Объект уничтожен");

}

\*\*Класс может иметь только один деструктор.

Деструкторы не могут быть унаследованы или перегружены.

Деструкторы невозможно вызвать. Они запускаются автоматически.

Деструктор не принимает модификаторы и не имеет параметров.

1. **Класс и методы System.Object.**

В C# все классы неявно наследуются от базового класса System.Object. Это означает, что каждый класс в C# наследует методы, определенные в System.Object. Давайте рассмотрим основные методы этого класса:

ToString():

Возвращает строковое представление объекта.

По умолчанию возвращает имя типа объекта.

Для классов - выводит полное название класса с указанием пространства имен, в котором определен этот класс.

Equals(object obj):

Определяет, равен ли текущий объект другому объекту.-проверка на тождество а не равенстов

По умолчанию сравнивает ссылки на объекты.

GetHashCode():

Возвращает хэш-код для текущего объекта.

Используется в хэш-таблицах и других структурах данных.

Рекомендуется переопределять вместе с методом Equals.

public override int GetHashCode()

{

return name.GetHashCode() ^ age.GetHashCode();

}

GetType():

Возвращает объект Type, представляющий тип текущего экземпляра.

Clone():

Создает поверхностную копию текущего объекта.

Копирует значения полей, но не копирует объекты, на которые указывают ссылки.

Метод Finalize() в C# используется для выполнения очистки ресурсов перед удалением объекта из памяти.

1. **Статические методы и статические конструкторы класса.**

Статические методы принадлежат классу, а не экземпляру класса. Это означает, что вы можете вызывать статические методы без создания экземпляра класса. Статические методы объявляются с использованием ключевого слова static

public class MathUtilities

{

public static int Add(int a, int b)

{

return a + b;

}

}

int result = MathUtilities.Add(5, 3); // Вывод: 8

отсутствует ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта

в методе static допускается непосредственный вызов только других методов типа static

**для метода static непосредствен**но доступными оказываются только другие данные типа static, определенные в его классе

Конструктор экземпляра инициализирует данные экземпляра

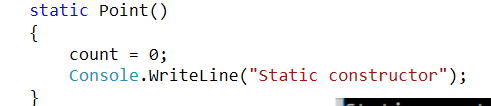
**конструктор класса (типа)— данные класса.**

Свойства:

закрытые автоматически

не имеет параметров

нельзя вызвать явным образом (вызываются до создания первого экземпляра объекта или до вызова любого статического метода).



1. **Статические классы. Методы расширения и правила их определения.**

Статический класс свойства

прямой потомок System.Object

экземпляры такого класса создавать запрещено

не должен реализовывать никаких интерфейсов (не вызвать)

нельзя использовать в качестве поля, параметра метода или локальной переменной

от него запрещено наследовать

все элементы такого класса должны явным образом объявляться с модификатором static

может иметь статический конструктор

**Компилятор не создает автоматически конструктор по умолчанию**

**Методы расширения** (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса.

Методы расширения определяются как статические методы в статическом классе, но вызываются как обычные методы экземпляра.

**Правила определения методов расширения**:

**Статический метод**: Метод расширения должен быть статическим.

**Статический класс**: Метод расширения должен быть определен в статическом классе.

**Первый параметр с ключевым словом** this: Первый параметр метода расширения указывает тип, который расширяется, и должен быть помечен ключевым словом this.

public static class StringExtensions

{

public static bool IsNullOrEmpty(this string str)

{

return string.IsNullOrEmpty(str);

}

}

)

ПРАВИЛА

1) Методы расширения должны быть объявлены в статическом необобщенном классе (первого уровня)

2) this перед первым аргументом и только оди

3)надо помнить, что метод расширения никогда не будет вызван, если он имеет ту же сигнатуру, что и метод, изначально определенный в типе.

1. **Анонимные типы.**

позволяют создать объект с некоторым набором свойств без определения класса (тип в одном контексте или один раз).

Используется в

Language Integrated Query, LINQ



· **Неизменяемость**: Свойства анонимных типов являются только для чтения, то есть их значения нельзя изменить после создания объекта.

· · **Вывод типов**: Компилятор автоматически выводит типы свойств на основе присваиваемых значений.

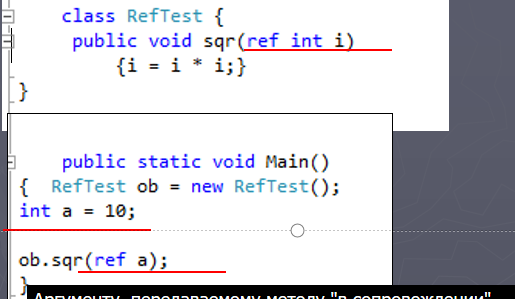
· · **Поддержка вложенных типов**: Анонимные типы могут содержать другие анонимные типы.

**Локальная область видимости: Анонимные типы могут использоваться только в пределах метода или блока кода, где они были созданы.**

1. **Модификаторы параметров - ref , out, params. Необязательные и именованные аргументы.**

ref заставляет С# организовать вместо вызова по значению вызов по ссылке

Аргументу, передаваемому методу "в сопровождении" модификатора ref, должно быть присвоено значение до вызова метода.

4

OUT можно использовать для передачи значения из метода

out-параметр "поступает" в метод без начального значения, но метод (до своего завершения) обязательно должен присвоить этому параметру значение

public void GetValues(out int x, out int y)

{

x = 10;

y = 20;

}

int a, b;

GetValues(out a, out b);

Console.WriteLine($"a: {a}, b: {b}"); // Вывод: a: 10, b: 20

params:

Модификатор params позволяет передавать переменное количество аргументов в метод.

Параметр с модификатором params должен быть последним в списке параметров метода.

public void PrintNumbers(params int[] numbers)

{

foreach (int number in numbers)

{

Console.WriteLine(number);

}

}

PrintNumbers(1, 2, 3, 4, 5); // Вывод: 1 2 3 4 5

### Необязательные аргументы

Необязательные аргументы позволяют задавать значения по умолчанию для параметров метода. Если аргумент не передан, используется значение по умолчанию.

csharp

public void DisplayMessage(string message, int repeatCount = 1){

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

{

Console.WriteLine(message);

}}

DisplayMessage("Hello"); // Вывод: HelloDisplayMessage("Hello", 3); // Вывод: Hello Hello Hello

### Именованные аргументы

Именованные аргументы позволяют указывать значения аргументов по имени параметра, что улучшает читаемость кода и позволяет передавать аргументы в любом порядке.

csharp

public void DisplayInfo(string name, int age, string city){

Console.WriteLine($"Name: {name}, Age: {age}, City: {city}");}

DisplayInfo(age: 25, name: "Alice", city: "New York"); /

1. **Перегрузка методов и операторов. Правила перегрузки операторов.**

Спецификация CLR требует, чтобы перегруженные операторные методы были

1) открытыми и статическими

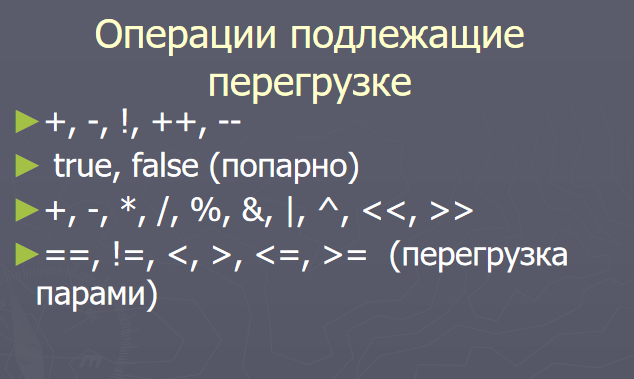
1. **тип одного из параметров или возвращаемого значения совпадал с типом, в котором определен операторный метод**

Перегрузка методов

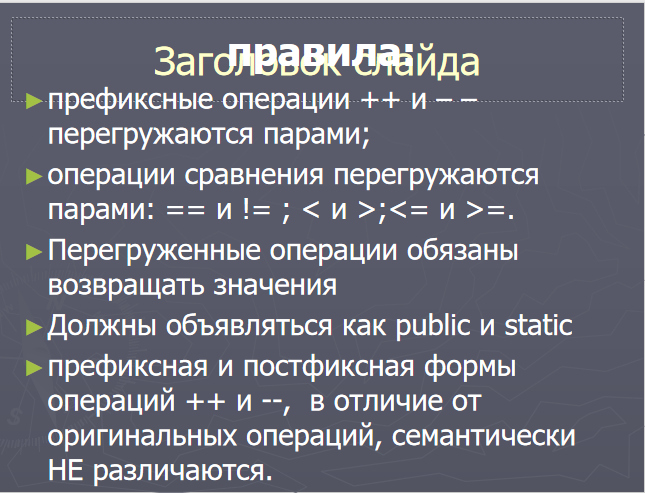
Перегрузка методов позволяет создавать несколько методов с одним и тем же именем, но с разными параметрами. Это позволяет использовать одно имя метода для выполнения различных задач в зависимости от переданных аргументов.

public int Add(int a, int b) { return a + b; } // Метод для сложения трех целых чисел

public int Add(int a, int b, int c) { return a + b + c; }



**public static Complex operator +(Complex c1, Complex c2) { return new Complex(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary); }**

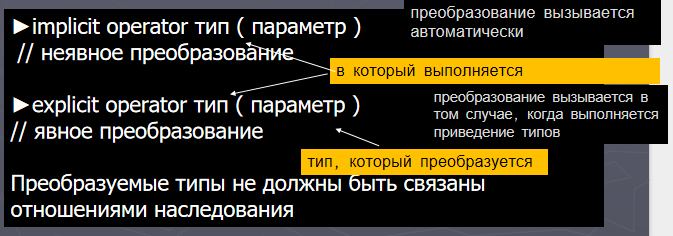


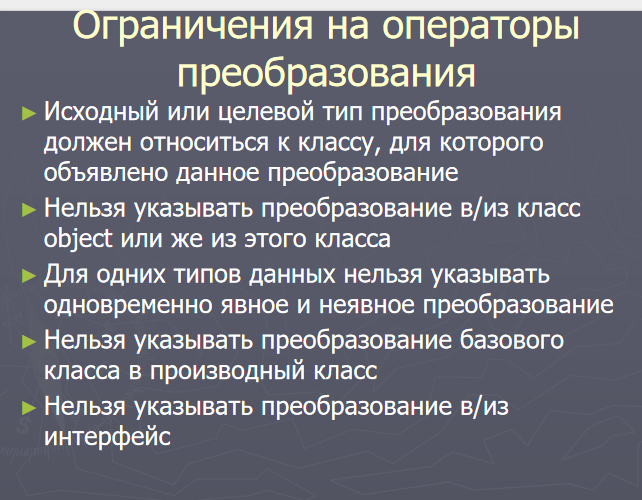
если перегружаются операторы == и !=, то для этого требуется переопределить методы Object.Equals() и Object.GetHashCode().

1. **Операции преобразования типа. Явная и неявная форма. Ограничения.**

преобразует объект исходного класса в другой тип

явная и неявна форма - будет ли этот алгоритм выполняться неявно или необходимо будет явным образом указывать необходимость соответствующего преобразования.





public class Celsius

{

public double Temperature { get; set; }

public static explicit operator Celsius(double temp)

{

return new Celsius { Temperature = temp };

}

}

Celsius c = (Celsius)37.0; // Явное преобразование double в Celsius

Console.WriteLine(c.Temperature); // Вывод: 37.0

public class Fahrenheit

{

public double Temperature { get; set; }

public static implicit operator Fahrenheit(double temp)

{

return new Fahrenheit { Temperature = temp };

}

}

Fahrenheit f = 98.6; // Неявное преобразование double в Fahrenheit

Console.WriteLine(f.Temperature); // Вывод: 98.6

### Явное преобразование (Explicit Conversion)

Явное преобразование требует явного указания типа, в который нужно преобразовать значение. Это делается с помощью оператора приведения (type).

Пример явного преобразования:

csharp

double d = 9.78;int i = (int)d; // Явное преобразование double в intConsole.WriteLine(i); // Вывод: 9

Parse Convert.ToInt32

### Неявное преобразование (Implicit Conversion)

Неявное преобразование происходит автоматически, когда нет потери данных и преобразование безопасно. Это преобразование не требует явного указания типа.

Пример неявного преобразования:

csharp

int i = 10;double d = i; // Неявное преобразование int в doubleConsole.WriteLine(d); // Вывод: 10.0

1. **Вложенные типы. Вложенные объекты**

объединения

1. **Правила наследования C#.**

объединения

1. **Сокрытие имен при наследовании. Обращение к срытым членам**

объединения

1. **Использование операций is и as**

объединения

1. **Полиморфизм. Виртуальные методы, свойства и индексаторы. Правила переопределения.**

объединения

1. **Понятие раннего и позднего связывания.**

объединения

1. **Абстрактные классы и методы. Бесплодные классы.**

объединения

1. **Структуры в C#.**

объединения

1. **Интерфейсы. Свойства интерфейсов. Реализация интерфейсов.**

объединения

1. **Явная и неявная реализация интерфейсов. Работа с объектами через интерфейсы.**

объединения

**31. Ковариантность интерфейсов. Контравариантность интерфейсов**

**32. Стандартные интерфейсы .NET. Назначение и применение.**

**33. Исключительные ситуации. Генерация и повторная генерация исключений.**

**34. Исключительные ситуации. Варианты обработки исключений. Фильтры исключений**

**35. Обобщения (generics). Свойства обобщений.**

**36. Концепция ограничений обобщений. Статические члены обобщений.**

**37. Делегаты. Определение, назначение и варианты использования. Обобщенные делегаты.**

**38. Анонимные функции. Лямбда-выражения.**

**39. Обобщённые делегаты .NET. Action, Func, Predicate**

**40. События и делегаты.**

**41. Стандартные коллекции .NET. Типы коллекций.**

**42. Стандартные интерфесы коллекций.**

**43. IEnumerable и IEnumerator**

**44. LINQ to Objects. Синтаксис. Форма. Возврат результата. Грамматика выражений запросов. Отложенные и неотлаженные операции.**

**45. LINQ to Objects. Операции Where, Select, Take, OrderB, Join, GroupBy**

**46. Рефлексия. System Type.**

**47. Классы для работы с файловой системой.**

**48. Синтаксическая конструкция using. Чтение и запись файлов. Потоковые классы.**

**49. Классы адаптеры потоков.**

**50. Сериализация. Форматы сериализации.**

**51. Сериализация контрактов данных. интерфейс ISerializable.**

**52. Атрибуты. Создание собственного атрибута.**

**53. Процесс. Домен приложений. Поток выполнения.**

**54. Создание потоков , классы приоритетов. Состояния потоков**

**55. Синхронизация потоков. Lock. Monitor. Мutex. Semaphore**

**56. Библиотека параллельных задач TPL. Класс Task. Состояние задачи.**

**57. Способы создания Task. Возврат результата. Отмена выполнения задач. Продолжения.**

**58. Параллелизм при императивной обработке данных. Класс Parallel**

**59. Асинхронные методы. async и await**

**60. Проектирование отношений. Агрегация, композиция и ассоциация**

**61. Антипаттерны проектирования. Рефакторинг. Методы рефакторинга.**

**62. Чистый код. Требования к именам, функциям, форматированию.**

**63. Чистый код. Требования к классам и объектам.**