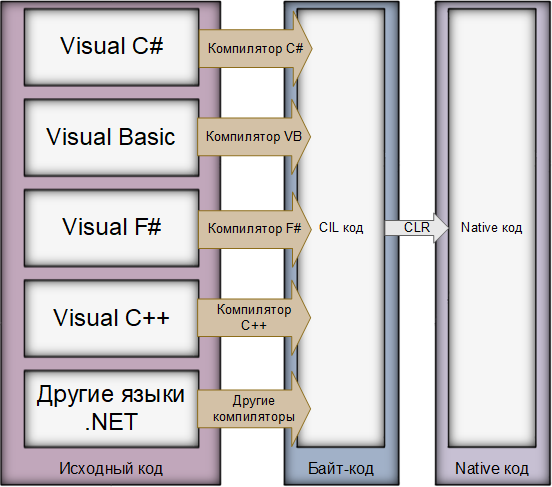
**1. Состав. NET Framework. Структура среды выполнения CLR**

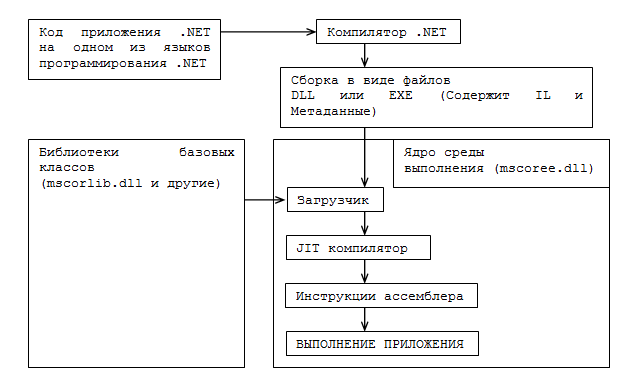
**Состав .Net Framework**

1)Общеязыковая исполняющая среда (CLR) Common Language Runtime) Виртуальная Машина. Обеспечивает выполнение сборки (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация)2)Библиотека классов (FCL). (.NET Framework Class Library) спецификации объектно-ориентированной библиотеки классов, интерфейсов и системы типов (типов-значений)соответствующая CLS

3)BSL– Base Class Library

**Состав CLR**

Функции:1)управление памятью;2)выполнение потоков;3)выполнение кода;4)проверка безопасности кода;5)управление системными службами и т.д



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

**Структура PE:**

1)Заголовок PE32 или PE32+ 4)Метаданные

2)тип файла: GUI, CUI, DLL; 5)Код Intermediate Language (IL)

3)Заголовок CLR

**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).

**Сборка (assembly)** — 1) это абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов.2) дискретная единица многократно используемого кода внутри CLR**Исполнение сборки:**

**JIT-компилятор** (Just-In-Time)1)**CLR** ищет типы данных и загружает во внутренние структуры 2)Для каждого метода CLR заносит адрес внутренней CLR функции JITCompiler3)**JITCompiler** ищет в метаданных соответствующей сборки IL-код вызываемого метода, проверяет и компилирует IL-код в машинные команды4)Они хранятся в динамически выделенном блоке памяти. 5)**JITCompiler** заменяет адрес вызываемого метода адресом блока памяти, содержащего готовые машинные команды6)**JITCompiler** передает управление коду в этом блоке памяти. **CLI(Common Language Infrastructure) –** это стандарт, который позволяет работать различным ЯП вместе.

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.**3. 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.

|  |  |  |
| --- | --- | --- |
| Тип С# | Размер в битах | Тип System(FCL) |
| sbyte | 8 | System.Sbyte |
| short | 16 | System.Int16 |
| int | 32 | System.Int32 |
| long | 64 | System.Int64 |
| byte | 8 | System.Byte |
| ushort | 16 | System.Uint16 |
| uint | 32 | System.Uint32 |
| ulong | 64 | System.UInt64 |
| char | 16 | System.Char |
| bool | 8 | System.Boolean |
| float | 32 | System.Single |
| double | 64 | System.Double |
| decimal | 128 | System.Decimal |
| string | – | System.String |
| object | – | System.Object |

**Ссылочные типы** – String и Object.

**Значимые** – примитивные, структуры, перечисления.

**4. 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.

**Распаковка (unboxing)** – получение указателя на исходный значимый тип (поля данных), содержащийся в объекте.

**Nullable-тип** – позволяет примитивным типам присваивать значение null.

Value - значение объектаHasValue: возвращает true, если объект хранит некоторое значение, и false, если объект равен null.**Null-объединение(??)** – возвращает левый операнд, если этот операнд не равен null иначе возвращается правый операнд.

**5. 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.**Создание строки:**

1)char[] a = { '0', '0', '0' }; // создание массива символов:2)string s; // инициализация отложена3)string t = "qqq"; // инициализация строковым литералом4)string u = new string(' ', 20); // с пом. конструктора5)string v = new string(a); // создание из массива символов**Операции для строк:**

1)присваивание (=);2)проверка на равенство содержимого (==);3)проверка на неравенство (!=);4)обращение по индексу ([]);5)сцепление (конкатенация) строк (+)6)<,>, >=,<= - сравнивают ссылки!!!!!!!!7)Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.8)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. 9)Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

**Пустая строка** — экземпляр объекта System.String, содержащий 0 символов: string s = ""; Для пустых строк можно вызывать методы. Cтроки 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.

**формат**{index[,alignment][:formatString]}“C” – для валюты, “D” – для десятичных чисел, “E” – для научной нотации, “F” – для нотации с фиксированной точкой {0,10:E} интерфейсы IFormatter и ICustomFormatter

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

Можно использовать 2 ключевых слова для неявно типизированных переменных: var и dynamic. Их отличия:

1)Для dynamic тип определяется на этапе выполнения, для var на этапе компиляции

2)Var может применяться только для объявления локальной переменной

3) Var не позволяет множественную инициализацию

Неявная типизация в C# позволяет компилятору автоматически определять тип переменной на основе присвоенного значения. Это упрощает код и делает его более читаемым. Основной механизм для неявной типизации в C# реализован с помощью ключевого слова var.

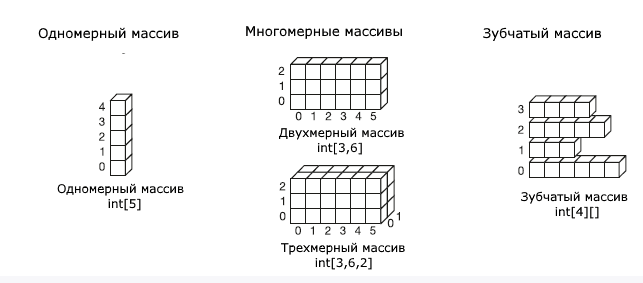
**7. Массивы C# одномерные, прямоугольные и ступенчатые.**

**Одномерные массивы:**

тип[] имя; тип[] имя = new тип [ размерность ]; тип[] имя = { список инициализаторов }; тип[] имя = new тип [] { список инициализаторов }; тип[] имя = new тип [ размерность ] { список инициализаторов };**Прямоугольные массивы:**

тип[,] имя; тип[,] имя = new тип [ разм\_1, разм\_2 ]; тип[,] имя = { список инициализаторов }; тип[,] имя = new тип [,] { список инициализаторов }; тип[,] имя = new тип [ разм\_1, разм\_2 ] { список инициализаторов }; **Ступенчатые массивы:**

тип[][] имя;



**8. Понятие кортежей. Свойства, создание**

**Кортежи (tuple)** комбинируют объекты различных типов (от одного до восьми).

ValueTuple < string, int> student = ("Olga", 19);(string, string, int) namesAndAge = ("Olga", "Krol", 22);var names = (first: "Никита", second: "Грицевич");string firstName = names.first;string secondName = names.second;(string first, string second) names2 = ( "Никита", "Крицевич");Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>**Свойства:**1)создается один раз и остается неименным (все свойства доступны только для чтения)2)позволяют использовать методы CompareTo, Equals, GetHashCode и ToString, свойство Size3)реализуют интерфейсы IStructuralEquatable, IStructuralComparable и IComparable (можно сравнивать)

**9. Принципы объектно-ориентированного программирования.**

**Объектно-ориентированное программирование** – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

**Принципы ООП:**

1)Инкапсуляция (Encapsulation);2)Наследование (Inheritance);3)Полиморфизм (Polymorphism);4)Абстракция данных(Abstraction). **Инкапсуляция** – механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования.

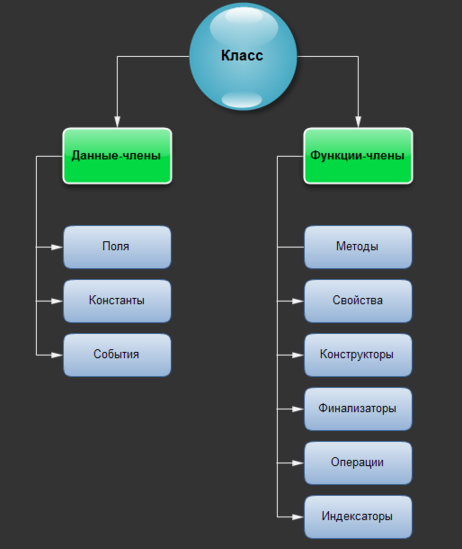
**Абстракция** подразумевает разделение и независимое рассмотрение интерфейса и реализации способ выделить набор наиболее важных атрибутов и методов и исключить незначимые.**Наследование** – процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта. Можно создавать иерархию классов.

**Полиморфизм** – позволяет 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.

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

**Класс** – описание множества объектов и выполняемых над ними действий.**Класс** – это некоторое абстрактное понятие, шаблон, по которому определяется форма объекта.

**Объект** – это физическая реализация класса(шаблона).

**Свойства** определяют методы записи и чтения.

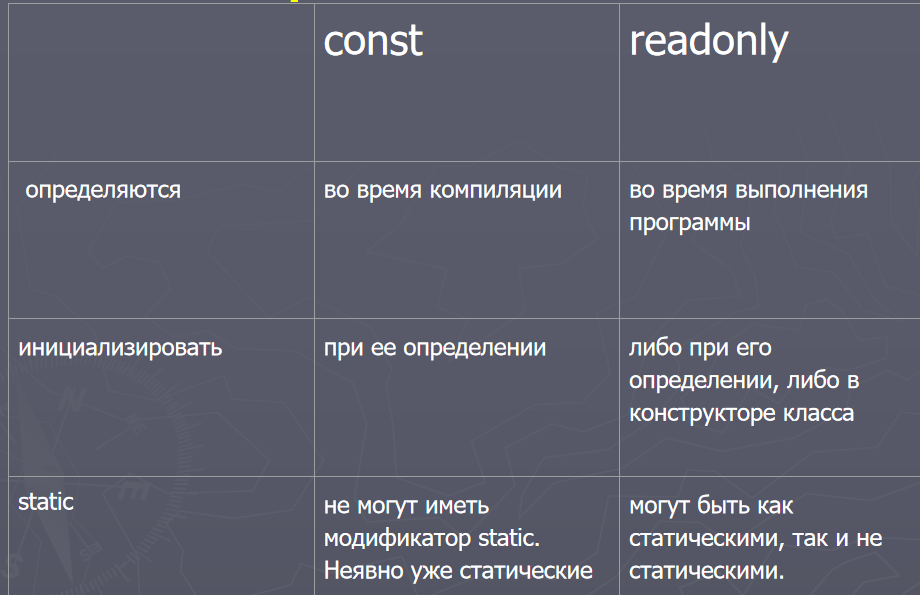
**Индексаторы** обеспечивают возможность доступа к элементам класса по их порядковому номеру.

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

**Класс** – описание множества объектов и выполняемых над ними действий.**Класс** – это некоторое абстрактное понятие, шаблон, по которому определяется форма объекта.

**Константы:**

1) компилятор сохраняет значение константы в метаданных модуля, константы можно определять только для таких типов, которые компилятор считает примитивными2) константы считаются не явно статическими, всегда связаны с типом, а не с экземпляром типа3) нельзя получать адрес константы и передавать ее по ссылке4) определять можем один раз 5)к моменту компиляции они должны быть определены.**Поля только для чтения:**

readonly - инициализация времени испол.1) Запись в поле разрешается при объявлении или в коде конструктора2) Инициализировать или изменять их значение в других местах нельзя, можно только считывать их значение.

**Конструкторы** — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа.С помощью **инициализатора** объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.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.

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

**Видимость типов:**

может быть открытым (public) или внутренним (internal).

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

**Класс** – описание множества объектов и выполняемых над ними действий.**Класс** – это некоторое абстрактное понятие, шаблон, по которому определяется форма объекта.

**Конструкторы** — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа.**Свойства конструкторов:**

1) имя такое же как и имя типа (класса )2) не имеет возвращаемого значения3) не наследуются4) нельзя применять модификаторы virtual, new, override, sealed и abstract5) для класса без явно заданных конструкторов компилятор создает конструктор по умолчанию (без параметров)6) для статических классов компилятор не создает конструктор по умолчанию7) может определяться несколько конструкторов, сигнатуры и уровни доступа к конструкторам обязательно должны отличаться8) можно явно заставлять один конструктор вызывать другой конструктор посредством зарезервированного слова this: **Статические конструкторы:**

1)не должны иметь модификатор доступа и не принимают параметров2)нельзя использовать ключевое слово this для ссылки на текущий объект класса и можно обращаться только к статическим членам класса3)нельзя вызвать в программе вручную. 4)Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам**Закрытый конструктор** используется в случаях, если вам необходимо контролировать создание объектов.

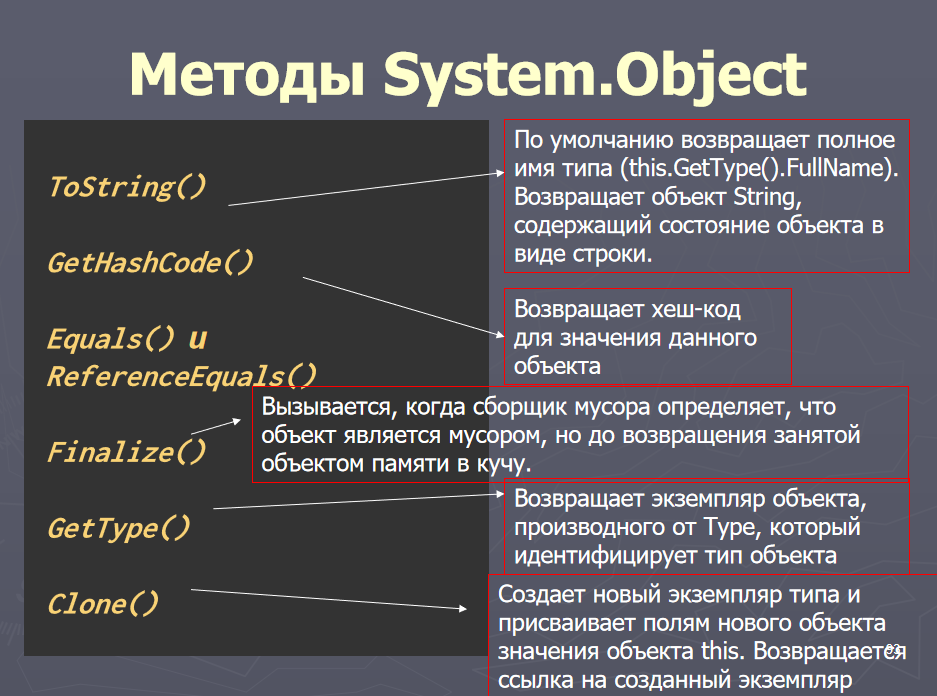
**Деструкторы:**

Вызываются непосредственно перед окончательным уничтожением объекта системой "сборки мусора", чтобы гарантировать четкое окончание срока действия объекта.~имя\_класса () { // код деструктора } **Свойства деструктора:**

1)Класс может иметь только один деструктор.2)Деструкторы не могут быть унаследованы или перегружены.3)Деструкторы невозможно вызвать. Они запускаются автоматически.4)Деструктор не принимает модификаторы и не имеет параметров.

**14. Класс и методы System.Object.**

**Класс** – описание множества объектов и выполняемых над ними действий.**Класс** – это некоторое абстрактное понятие, шаблон, по которому определяется форма объекта.



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

**Статические конструкторы:**

1)не должны иметь модификатор доступа и не принимают параметров2)нельзя использовать ключевое слово this для ссылки на текущий объект класса и можно обращаться только к статическим членам класса3)нельзя вызвать в программе вручную. 4)Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам**Статические члены класса:**

переменные и свойства, которые хранят состояние, общее для всех объектов класса, следует определять как статическиеметоды, которые определяют общее для всех объектов поведение, также следует объявлять как статические.

При использовании статических членов необязательно создавать экземпляр класса.

Для статических полей будет создаваться участок в памяти, который будет общим для всех объектов класса.**Свойства статических методов:**1)отсутствует ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта2)в методе static допускается непосредственный вызов только других методов типа static3)для метода static непосредственно доступными оказываются только другие данные типа static, определенные в его классе.

**Свойства статических конструкторов:**1)закрытые автоматически2)не имеет параметров3)нельзя вызвать явным образом (вызываются до создания первого экземпляра объекта или до вызова любого статического метода).

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

**Статический класс:**

1)прямой потомок System.Object2)экземпляры такого класса создавать запрещено3)не должен реализовывать никаких интерфейсов (не вызвать)4)нельзя использовать в качестве поля, параметра метода или локальной переменной5)от него запрещено наследовать6)все элементы такого класса должны явным образом объявляться с модификатором static8)может иметь статический конструктор9)Компилятор не создает автоматически конструктор по умолчанию**Методы расширения** (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса.

**Правила для методов расширения:**

1) Методы расширения должны быть объявлены в статическом необобщенном классе (первого уровня)2) this перед первым аргументом и только один3) надо помнить, что метод расширения никогда не будет вызван, если он имеет ту же сигнатуру, что и метод, изначально определенный в типе.

**17. Анонимные типы.**

позволяют создать объект с некоторым набором свойств без определения класса (тип в одном контексте или один раз). var someType = new {Name = "Anna"};

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

для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров: 1)по умолчанию- параметры-значения; 2)параметры-ссылки — ref; 3)выходные параметры-ссылки — out: 4)переменное количество — params (один и последний). **ref** заставляет С# организовать вместо вызова по значению вызов по ссылке.

Модификатор **out** подобен модификатору **ref** за одним исключением: его 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.

**Необязательные аргументы:**

1)позволяет определить используемое по умолчанию значение для параметра метода2)можно применять в конструкторах, индексаторах

3) должны указываться справа от обязательных

**Именованные аргументы:**

позволяет указать имя того параметра, которому присваивается его значение(в конструкторах, индексаторах или делегатах.)

значение аргумента присваивается параметру по его позиции в списке аргументов.

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

**Перегрузка методов:**

1)один и тот же метод, но с разным набором параметров. методы имеют разную сигнатуру, в которой совпадает только название метода.2)позволяет обращаться к связанным методам посредством одного, общего для всех имени.3)никакие два метода внутри одного и того же класса не должны иметь одинаковую сигнатуру**методы должны отличаться по:**1)Количеству параметров 2)Типу параметров3)Порядку параметров4)Модификаторам параметров(не включает тип значения, возвращаемого методом, не включает params-параметр)**отличие методов по возвращаемому типу или по имени параметров не является основанием для перегрузки.**

**Перегрузка операций:**

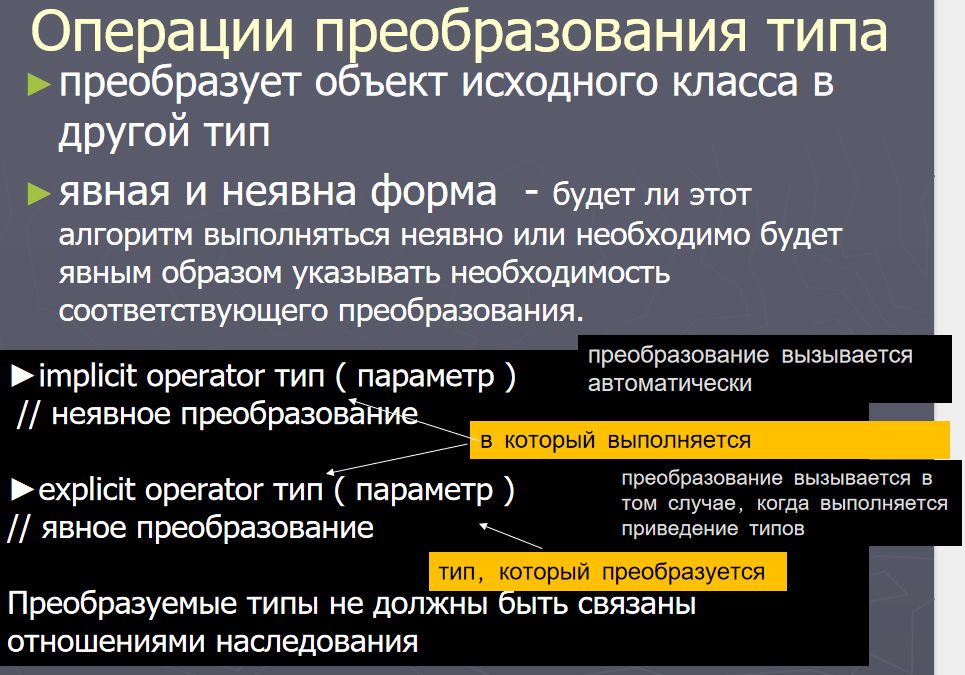
способ объявления новых операций для типаСпецификация CLR требует, чтобы перегруженные операторные методы были 1) открытыми и статическими2) тип одного из параметров или возвращаемого значения совпадал с типом, в котором определен операторный метод**Операции, которые подлежат перегрузке:**

1)+, -, !, ++, --2)true, false (попарно)3)+, -, \*, /, %, &, |, ^, <<, >>4)==, !=, <, >, <=, >= (перегрузка парами)**Операции не подлежащие перегрузке:**

1)[] (но есть индексатор)2)() (можно определить новые операторы преобразования)3)+=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= (но получаем автоматически в случае перегрузки бинарной операции)4)&&, || 5)=, ., ?:, ??, ->,=>, f(x), as, checked,unchecked, default, delegate, is, new, sizeof, typeof**Правила перегрузки операций:**

1)префиксные операции ++ и – – перегружаются парами;2)операции сравнения перегружаются парами: == и != ; < и >;<= и >=.3)Перегруженные операции обязаны возвращать значения4)Должны объявляться как public и static4)префиксная и постфиксная формы операций ++ и --, в отличие от оригинальных операций, семантически НЕ различаются.5) если перегружаются операторы == и !=, то для этого требуется переопределить методы Object.Equals() и Object.GetHashCode().

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



**Ограничения:**

1)Исходный или целевой тип преобразования должен относиться к классу, для которого объявлено данное преобразование2)Нельзя указывать преобразование в/из класс object или же из этого класса3)Для одних типов данных нельзя указывать одновременно явное и неявное преобразование4)Нельзя указывать преобразование базового класса в производный класс5)Нельзя указывать преобразование в/из интерфейса.

**21. Вложенные типы. Вложенные объекты**

**Вложенные типы** — это типы данных, которые определены внутри другого типа. Вложенные объекты позволяют организовать код и структуру данных более логично и удобно. В C# наиболее распространёнными примерами вложенных типов являются вложенные классы, структуры и перечисления.

**Вложенные классы**

Вложенные классы определяются внутри других классов. Они могут использовать члены внешнего класса и могут быть полезны для группировки логически связанных классов.

public class OuterClass

{

public class NestedClass

{

public void Display()

{

Console.WriteLine("Hello from NestedClass.");

}

}

}

**Вложенные структуры**

Вложенные структуры работают аналогично вложенным классам, но они представляют собой значимые типы.

public class OuterClass

{

public struct NestedStruct

{

public int Value;

public void Display()

{

Console.WriteLine($"Value: {Value}");

}

}

}

**Вложенные перечисления**

Перечисления также могут быть вложенными. Это полезно для определения связанных наборов констант внутри класса.

public class OuterClass

{

public enum NestedEnum

{

FirstValue,

SecondValue,

ThirdValue

}

}

**Вложенные объекты**

**Вложенные объекты** — это объекты, которые являются членами других объектов. Это позволяет создавать сложные структуры данных и модели.

public class Car

{

public Engine CarEngine { get; set; }

public Car()

{

CarEngine = new Engine();

}

public class Engine

{

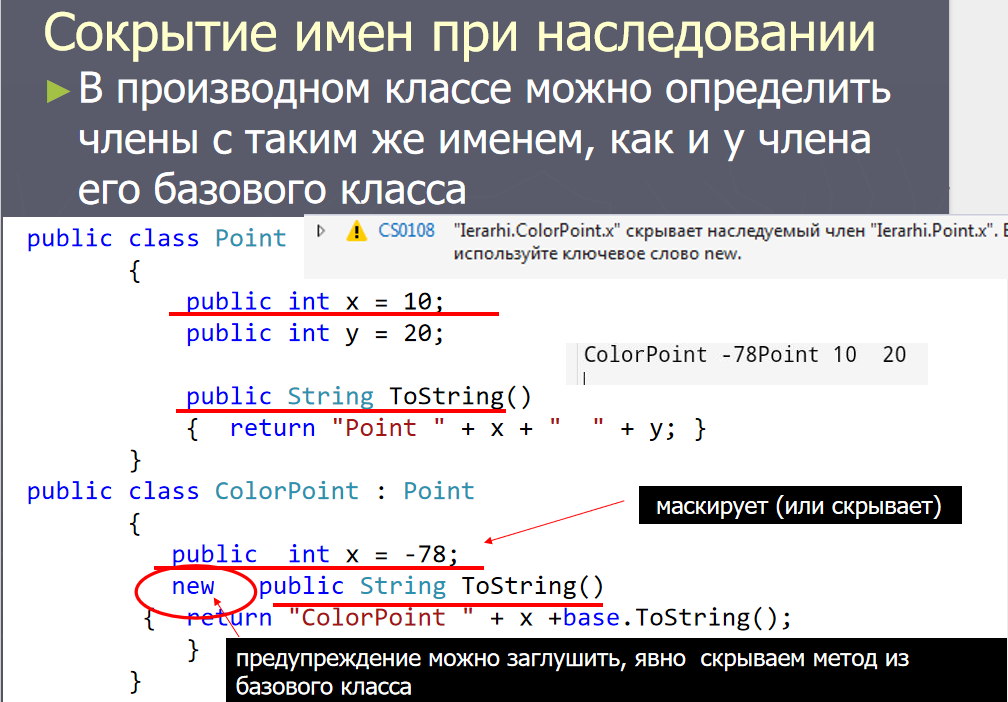
public int Horsepower { get; set; }

}

}

**22. Правила наследования C#.**

Правила наследования:1) В C# наследование всегда подразумевается открытым2) Запрещено множественное наследование классов (но не интерфейсов)3) наследуются все свойства, методы, поля и т.д., которые есть в базовом классе4) Производному классу доступны public, internal, protected и protected internalчлены базового класса (private – недоступны)5) не наследуются конструкторы базового класса (но можно вызвать)6) тип доступа к производному классу должен быть таким же, как и у базового класса или более строгим7) Ссылке на объект базового класса можно присвоить объект производного класса (но вызываются для него только методы и свойства, определенные в базовом классе.)

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



**24. Использование операций is и as**

**Операция is:**

Возвращает булевское значение, говорящее о том, можете ли вы преобразовать данное выражение в указанный типОператор **is** никогда не генерирует исключение.

**Операция as:**

позволяет преобразовывать тип в определенный ссылочный тип с применением следующего синтаксиса: операнд as <тип>**Выполняется:**

1)Если <операнд> имеет тип, заданный в <тип>. 2)Если <операнд>, может быть неявно преобразован в <тип>. 3)Если операнд <операнд>, может быть упакован в <тип>. as - никогда не генерирует исключение

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

**Полиморфизм:**

1)ключевой аспект объектно-ориентированного программирования 2)способность к изменению функций, унаследованных от базового класса**полиморфный интерфейс в базовом классе - набор членов класса, которые могут быть переопределены в классе-наследнике:**

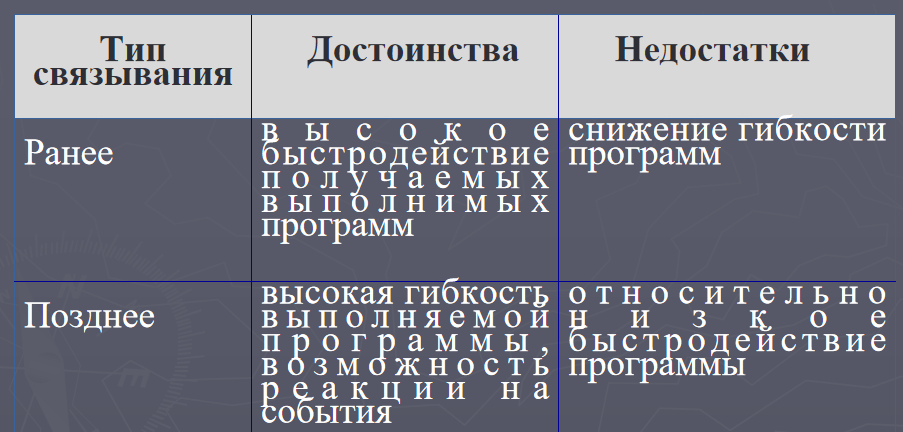
virtual public void A\_method() { }

**переопределение виртуального метода в производном классе:** override public void A\_method() { }

**Правила переопределения:**

1) Переопределенный виртуальный метод должен обладать таким же набором параметров, как и одноименный метод базового класса.2) не может быть static или abstract3) вызывается ближайший вариант, обнаруживаемый вверх по иерархии (многоуровневая)Если не virtual переопределять нельзя

**26. Понятие раннего и позднего связывания.**

**раннее связывание** – адрес функции назначается во время компиляции, и именно этот адрес используется при вызове функции (статический)**позднее связывание** (только для методов классов) – связанное с формированием кода на этапе выполнения. Вызов метода происходит на основании типа объекта, а не типа ссылки на базовый класс (динамический)**Необходимые условия для реализации позднего связывания:**1)классы должны образовывать иерархию наследования;2)в классах должны быть методы с одинаковой сигнатурой. Элементы (методы) производных классов должны перекрывать (override) соответствующие элементы (методы) базовых классов;3)элементы (методы) класса должны быть виртуальными, то есть должны быть обозначены ключевыми словами virtual, override.

**27. Абстрактные классы и методы. Бесплодные классы.**

**Бесплодные классы** – класс, от которого наследовать запрещается.

**Абстрактный класс:**

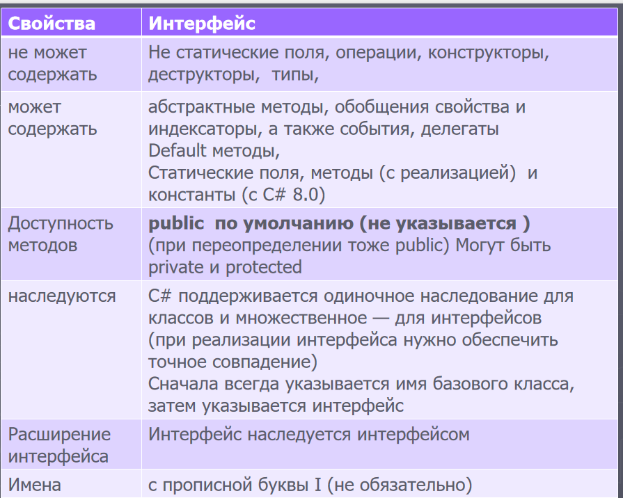
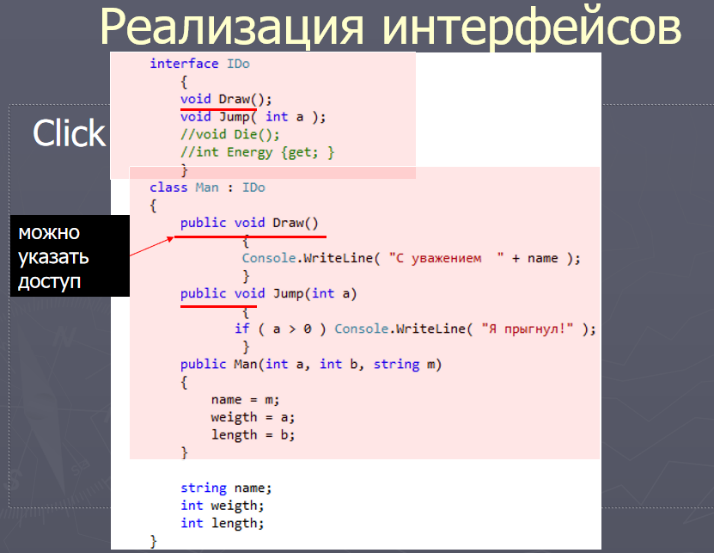
1)Служит только для порождения потомков - предоставляют базовый функционал для классов-наследников.2)Задает интерфейс для всей иерархии3)Может содержать и полностью определенные методы, переменные, конструкторы, свойства4)Создавать объект абстрактного класса нельзя!!!!!!! (ссылку можно)если класс имеет хотя бы одно абстрактное свойство или метод, то он должен быть определен как абстрактный.**абстрактными могут быть:**1)Методы2)Свойства3)Индексаторы4)События**Свойства абстрактных методов:**

1) абстрактные методы автоматически виртуальные (virtual не ставится)2) абстрактные методы не используются со static3) А.К. может быть параметром метода - полиморфные методы

**28. Структуры в C#.**

1) struct2) Может иметь конструктор c парам.3) до C# 10 нельзя определить конструктор, используемый по умолчанию (конструктор без параметров). Он определяется для всех структур автоматически и не подлежит изменению4) Объект структуры может быть создан с помощью оператора new (или нет) или default5) размещение в стеке6) До C# 10  Нельзя инициализировать поля структуры при объявлении7) До C# 10  нет автоматической инициализации полей компилятором8) структуры не поддерживают наследование9) Могут реализовывать интерфейсы10) нельзя объявить деструктор (метод завершения) в типе структурыНазначение : повышении эффективности и производительности программ (тип значения) struct Time { private int hours = 0; // ошибка в ходе компиляции private int minutes; private int seconds; }

**29. Интерфейсы. Свойства интерфейсов. Реализация интерфейсов.**

позволяют определить требования к реализации (контракт)Задается набор абстрактных методов, свойств, событий и индексаторов, которые должны быть реализованы в производных классах

Интерфейс или класс может наследовать свойства нескольких интерфейсов, в этом случае предки перечисляются через запятую

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

**Неявная реализация интерфейсов** происходит, когда члены интерфейса реализуются как обычные методы или свойства класса. Это позволяет обращаться к ним напрямую через экземпляр класса.

public interface IExample

{

void Display();

}

public class Example : IExample

{

public void Display()

{

Console.WriteLine("This is a non-explicit implementation of the interface.");

}

}

**Явная реализация интерфейсов** используется, когда члены интерфейса реализуются с использованием имени интерфейса. Это позволяет избежать конфликтов имен, если класс реализует несколько интерфейсов с одинаковыми методами.

public interface IExample

{

void Display();

}

public interface IAnotherExample

{

void Display();

}

public class Example : IExample, IAnotherExample

{

void IExample.Display()

{

Console.WriteLine("This is an explicit implementation for IExample.");

}

void IAnotherExample.Display()

{

Console.WriteLine("This is an explicit implementation for IAnotherExample.");

}

}

Переменной ссылки на интерфейс доступны только методы, объявленные в ее интерфейсе.Класс наследует все методы своего предка (интерфейсы). Он может переопределить ( new), но обращаться к ним - через объект класса. Если использовать для обращения ссылку на интерфейс, вызывается не переопределенная версияесли интерфейс реализуется с помощью виртуального метода класса, после его переопределения в потомке любой вариант обращения (через класс или через интерфейс) приведет к одному и тому же результатуМетод интерфейса, реализованный явным указанием имени, объявлять виртуальным запрещается. При необходимости переопределить в потомках его поведение:Существует возможность повторно реализовать интерфейс, указав его имя в списке предков класса наряду с классом-предком, уже реализовавшим этот интерфейс



**31. Ковариантность интерфейсов. Контравариантность интерфейсов**

**Ковариантность** позволяет использовать более производные типы вместо базовых типов. Это применимо только к выходным параметрам (например, методам, которые возвращают значения).

public interface ICovariant<out T>

{

T GetValue();

}

public class Example : ICovariant<string>

{

public string GetValue()

{

return "Covariant Example";

}

}

ICovariant<object> covariant = new Example(); // Присваивание

object value = covariant.GetValue(); // Получение значения

**Контравариантность** позволяет использовать более общие типы вместо более производных. Это применимо только к входным параметрам (например, методам, которые принимают аргументы).

public interface IContravariant<in T>

{

void SetValue(T value);

}

public class Example

{

public void SetValue(object value)

{

Console.WriteLine("Contravariant Example: " + value);

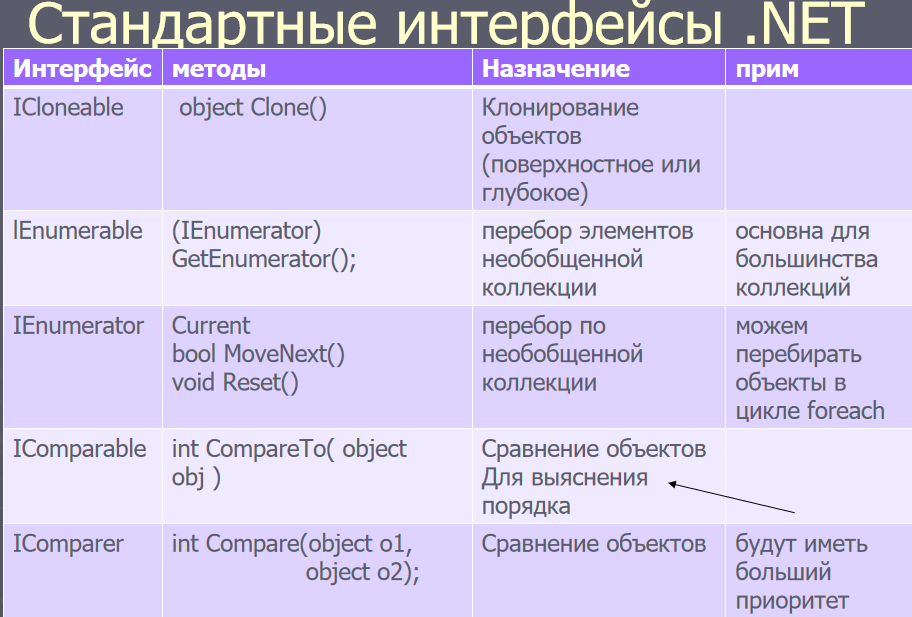
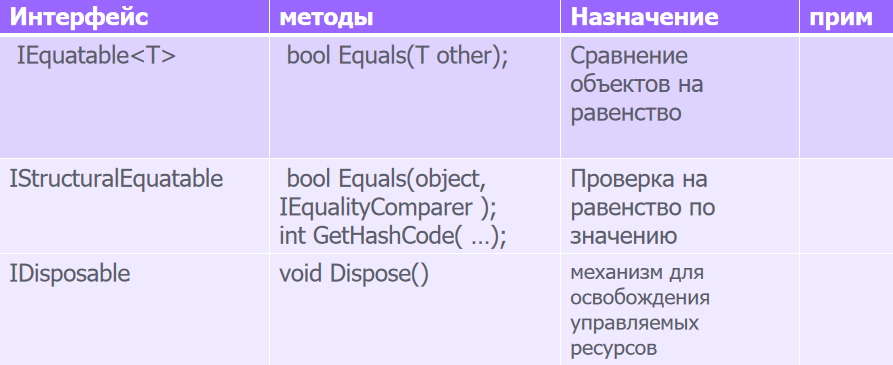
}

}

IContravariant<string> contravariant = new Example();

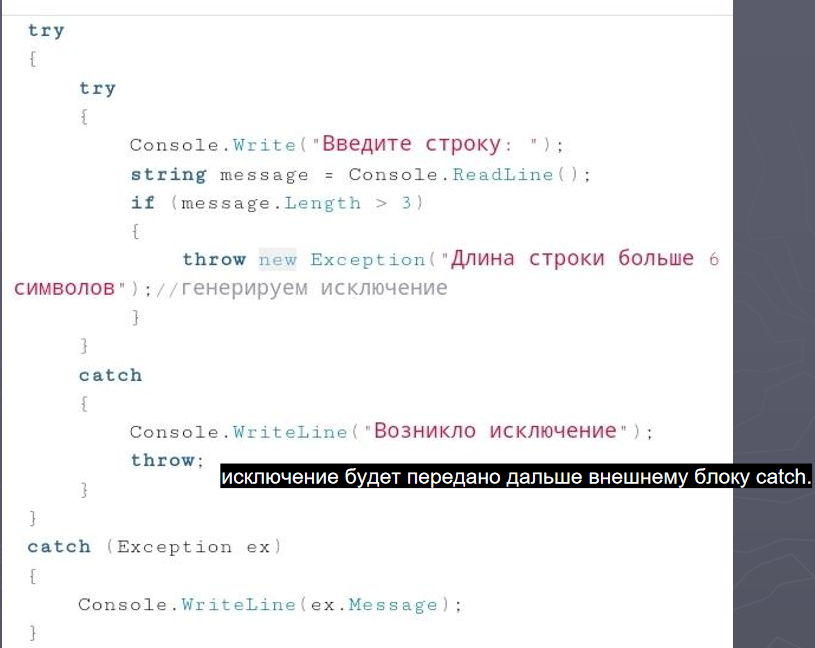
contravariant.SetValue("Contravariant Example"); // Передача значения

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

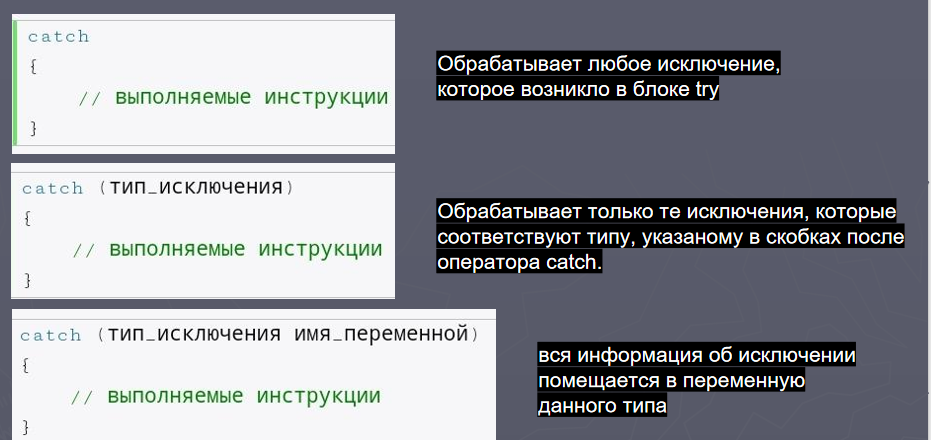
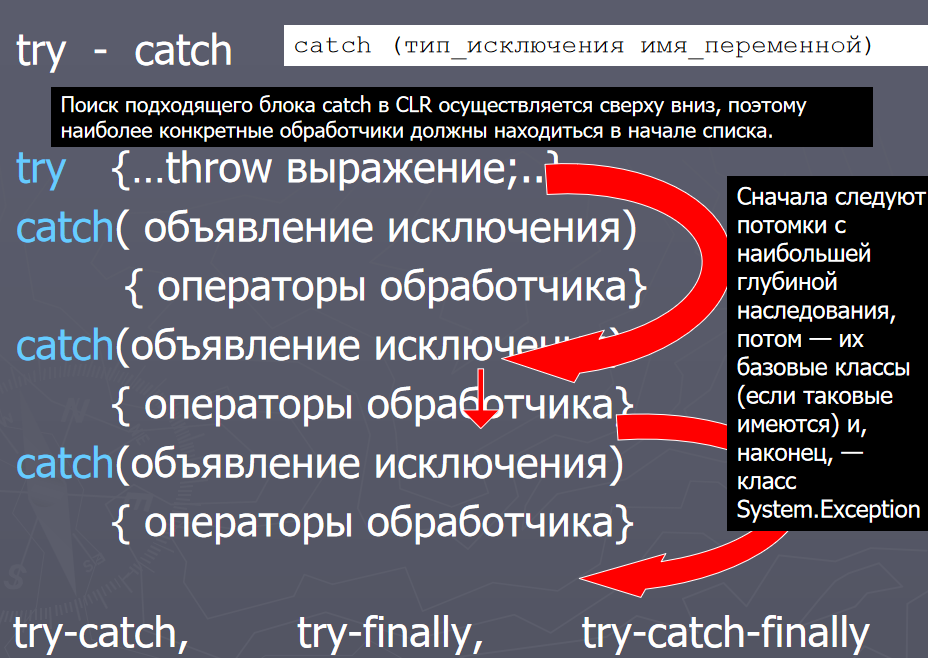


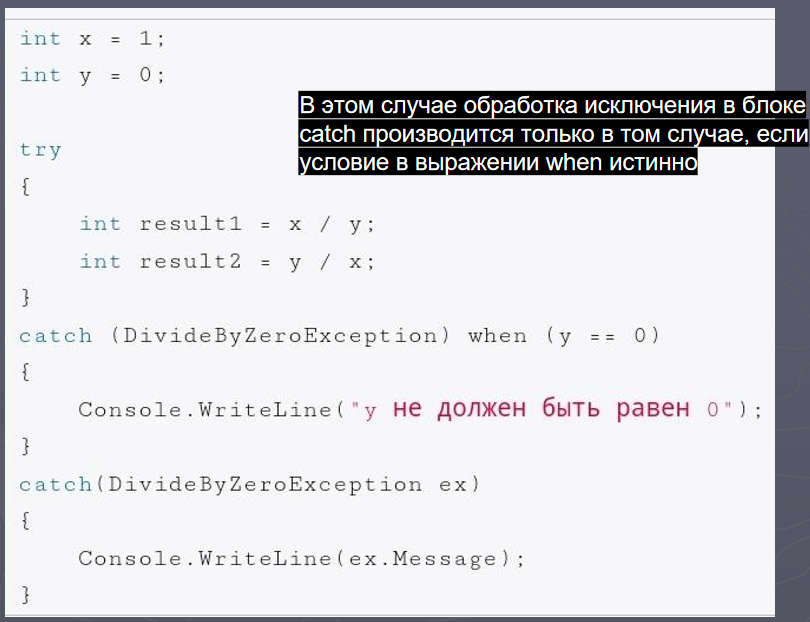
**33. Исключительные ситуации. Генерация и повторная генерация исключений.**

**Исключительная ситуация** – это состояние ошибки, обнаруженное в программе в ходе ее выполнения (деление на ноль, невозможность выделения памяти при создании нового объекта и т.д. )Генерируется оператором **throw**( <выражение>)аргумент - объект типа исключение**try** – контролируемый блок**throw** - генерация искл. ситуации внутри try**catch** – обработчики исключений, идут за try (несколько)**finally** - код, очищающий ресурсы и др. действия (выполняется всегда) (один на один try)**Повторная генерация исключений:**

создание нового объекта посредством повторного использования старого с помощью оператора throw без параметров. В этом виде оператор throw может использоваться исключительно в блоке catch

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

**Исключительная ситуация** – это состояние ошибки, обнаруженное в программе в ходе ее выполнения (деление на ноль, невозможность выделения памяти при создании нового объекта и т.д. )

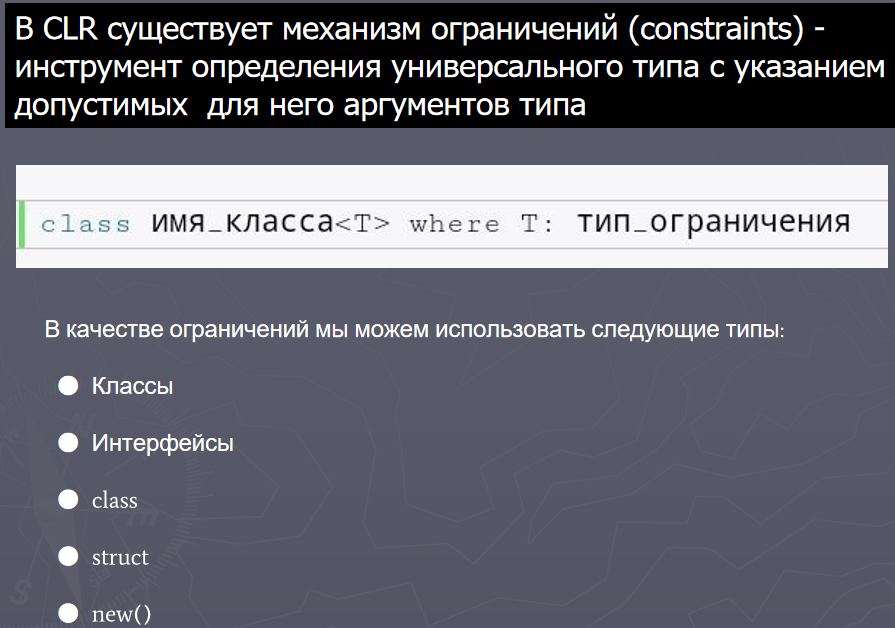
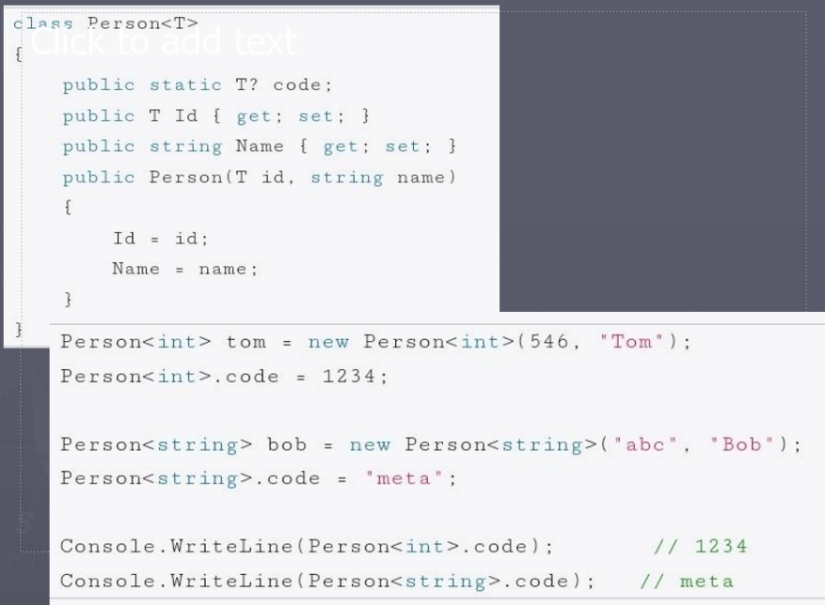
**Фильтр исключения** позволяет указать дополнительные условия, при которых используется обработчик исключения. Эти условия принимают форму булева выражения, перед которым ставится ключевое слово **when**.

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

**Обобщение** - параметризированный типОпределены для CLR – поддержка разных языковОткрытый тип закрытый тип Tlist<T> Tlist<int> System.Collections.Genericpublic class SuperArray<T> { T[] s; }**Свойства:**

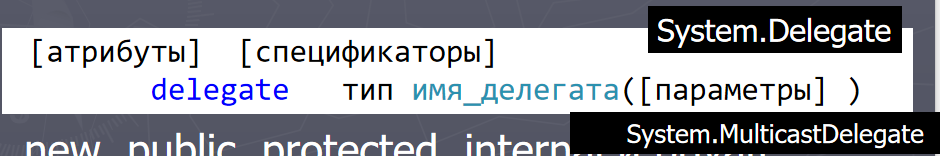
1)Универсальный тип может содержать другой универсальный тип2) Универсальные типы перегружаются на основе количества параметров

3) Универсальными могут быть классы, структуры , интерфейсы, делегаты, методыpublic void Method <R> (A<R> iA, B<R,T> iB)4) Могут содержать статические типы5) Доступность конструируемых типов определяется на основе пересечения доступности универсального типа и типа в списке аргументов 5) могут использовать несколько универсальных параметров одновременно6) поддерживает механизм ограничений**36. Концепция ограничений обобщений. Статические члены обобщений.**



в CLR размещает статические поля типа в самом объекте-типе , каждый закрытый тип имеет свои статические поля

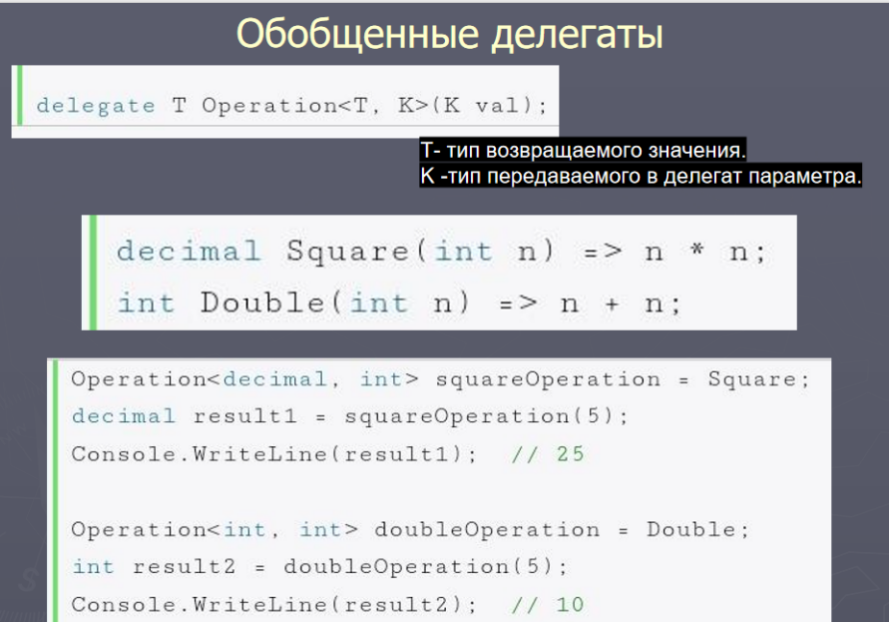
**37. Делегаты. Определение, назначение и варианты использования. Обобщенные делегаты.**

**Делегат** – это объект, предназначенный для хранения ссылок на методы(указатель на функцию C++) функции обратного вызова + без. типов1) используются для поддержки событий2) как самостоятельная конструкция языкаnew, public, protected, internal и private.

**Свойства:**

1)Тип данных (ссылочный)2)Наследовать от делегата нельзя3)Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса (в любом месте, где может быть определен класс)

4)может вызывать только такие методы, у которых тип возвращаемого значения и список параметров совпадают5)Может быть статический метод класса6)Имеет тот же синтаксис, что и вызов методa7)Если делегат хранит ссылки на несколько методов, они вызываются последовательно в том порядке, в котором были добавлены в делегат (цепочки (chaining))8) делегат можно вызывать как обычный метод9)делегаты могут быть параметрами методов**Назначение:**

1)возможности определять вызываемый метод не при компиляции, а динамически во время выполнения программы;2)обеспечения связи между объектами по типу «источник — наблюдатель»;3)создания универсальных методов, в которые можно передавать другие методы; 4)поддержки механизма обратных вызовов. 

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

**Анонимные функции:**

представляет собой безымянный кодовый блок, передаваемый конструктору делегата: 1)aнонимные методы 2)лямбда - выраженияПараметры должны соответствовать параметрам делегатаможет не содержать никаких параметровметод имеет доступ ко всем переменным, определенным во внешнем коде**Лямбда-выражение:**

упрощенная запись анонимных методовпараметр => выражение (список\_параметров) => выражение

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

void Action<in T1, in Т2, in ТЗ ….in T16 >  TResult Func<out TResult> TResult Func<in T1,……T16, out TResult>

**Action:**

представляет некоторое действие, которое ничего не возвращает, то есть в качестве возвращаемого типа имеет тип void:public delegate void Action<T>(T obj)**Func:**

Он возвращает результат действия и может принимать параметры

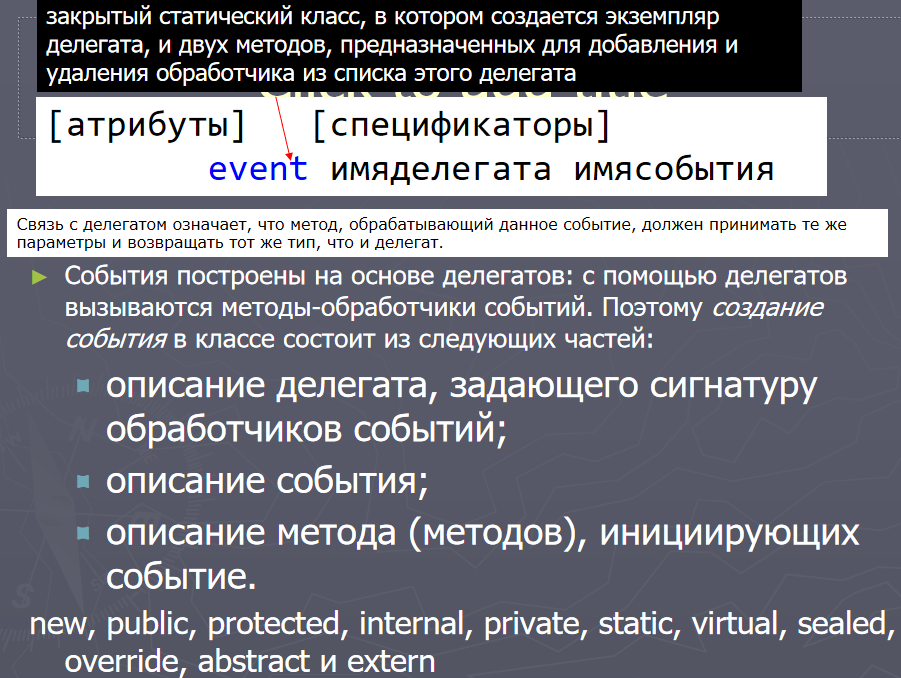
Func<in T1, in T2,...in T16, out TResult>()**Predicate:**

принимает один параметр и возвращает значение типа bool.используется для сравнения, сопоставления некоторого объекта T определенному условию. delegate<bool> Predicate<in T>(T obj)

**40. События и делегаты.**

**Событие:**

это элемент класса, позволяющий ему посылать другим объектам уведомления об изменении своего состояния1)модель «публикация — подписка»или паттерн «наблюдатель», класс, являющийся отправителем (sender) сообщения, публикует события, которые он может инициировать,

2) а другие классы, являющиеся получателями (receivers) сообщения, подписываются на получение этих событий.

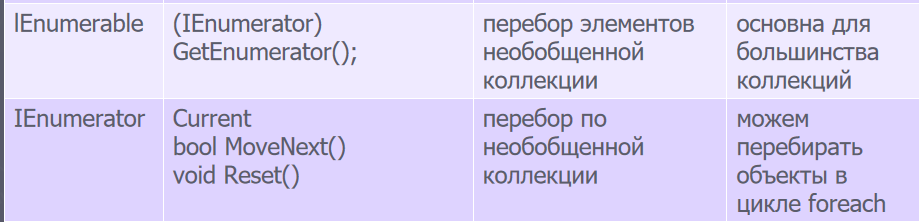
**41. Стандартные коллекции .NET. Типы коллекций.**

представляет собой совокупность объектов.в среду .NET Framework встроены коллекции, предназначенные для поддержки динамических массивов, связных списков, стеков, очередей и хеш-таблиц.1)необобщенные наличие разнотипных данных ссылки на данные типа object (не обеспечивают типовую безопасность) System.Collections2)обобщенные обеспечивают типовую безопасность System.Collections.Generic3)специальные System.Collections.Specialized с поразрядной организацией BitArray4)параллельные многопоточный доступ к коллекции System.Collections.ConcurrentКаждый класс коллекции оптимизирован под конкретную форму хранения данных и доступа к ним, каждый из них предоставляет специализированные методы

**42. Стандартные интерфесы коллекций.**

1)IEnumerable<T> для foreach, GetEnumerator() 2)IEnumerator<> 3)ICollection<T> Count, СоруТо(), Add(), Remove(), Clear()4)IList<T>Индексатор, Insert() , Remove()5)ISet<T>6)IDictionary<TKey, TValue>7)IComparer<T>8)ICollectionопределяет элементы

**43. IEnumerable и IEnumerator**



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

**Language Integrated Query – LINQ**Набор языковых и платформенных средств для написания структурированных и безопасных в отношении типов запросов к локальным коллекциям объектов и удаленным источником данным (базы данных, документы XML и т.д.)1)LINQ to Objects – библиотеки для обработки коллекций объектов в памяти,2)LINQ to SQL – библиотеки для работы с базами данных, LINQ to XML 3)LINQ to Entity**Операции запросов:**

1)отложенные операции 2)не отложенные операции**Возврат**:IEnumerable<T> или var **Код** :

1)именованные методы2)анонимные методы 3)лямбда-выражения **Форма:**

1)Выражения запросов (Операторы запросов LINQ)2)Стандартная точечная нотации C# с вызовом методов на объектах и классах (Методы расширений LINQ)Синтаксис выражений запросов поддерживается : Where, Select, SelectMany, Join, GroupJoin, GroupBy, OrderBy, ThenBy, OrderByDescending и ThenByDescending.**Грамматика выражений запросов:**

1) Начало - from2) 0..\* from, let или where. 3) orderby,  ascending  или descending 4) select или group.5) конструкции into, join, или повторение с п.2. **Отложенные операции:**

LINQ-выражение не выполняется, пока не будет произведена итерация или перебор по выборке, например, в цикле foreach1)После определения запроса он может выполняться множество раз.2)До выполнения запроса источник данных может изменяться.3)переменная запроса сама по себе не выполняет никаких действий и не возвращает никаких данных. Она только хранит набор команд, которые необходимы для получения результатов. То есть выполнение запроса после его создания откладывается. Само получение результатов производится при переборе в цикле foreach.**Не отложенные операции:**

методы, которые возвращают одно атомарное значение или один элемент или данные типов Array, List и Dictionary.

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

**Language Integrated Query – LINQ**Набор языковых и платформенных средств для написания структурированных и безопасных в отношении типов запросов к локальным коллекциям объектов и удаленным источником данным (базы данных, документы XML и т.д.)По типу обра1)LINQ to Objects – библиотеки для обработки коллекций объектов в памяти,2)LINQ to SQL – библиотеки для работы с базами данных, LINQ to XML 3)LINQ to Entity**Where:** определяет фильтр выборки**Select:** определяет проекцию выбранных значений**Take:** выбирает определенное количество элементов**OrderBy:** упорядочивает элементы по возрастанию**Join:** соединяет две коллекции по определенному признаку**GroupBy:** группирует элементы по ключу

**46. Рефлексия. System Type.**

**Рефлексия** – процесс выявления типов во время выполнения приложенияSystem.Reflection**System.Type:**

- класс, позволяет получить информацию о членах типа.представляет изучаемый тип, инкапсулируя всю информацию о нем.его свойства:1)**Name** возвращает имя типа2)**Assembly** возвращает название сборки, где определен тип3)**Namespace** возвращает название пространства имен, где определен тип4)**IsArray** возвращает true, если тип является массивом5)**IsClass** возвращает true, если тип представляет класс6)**IsEnum** возвращает true, если тип является перечислением7)**IsInterface** возвращает true, если тип представляет интерфейс8) FindMembers()  - воз. массив объектов MemberInfo данного типа9)GetConstructors() - конструкторы данного типа в виде набора объектов ConstructorInfo10)GetEvents() - события данного типа в виде массива объектов EventInfo11)GetFields() - поля данного типа в виде массива объектов FieldInfo12)GetInterfaces() - реализуемые данным типом интерфейсы в виде массива объектов Type13)GetMembers() -члены типа в виде массива объектов MemberInfo14)GetMethods() - методы типа в виде массива объектов MethodInfo15)GetProperties() - свойства в виде массива объектов PropertyInfo

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

**DriveInfo** – для представления диска GetDrives() - возвращает имена всех логических дисков компьютера.**Directory(статический) и DirectoryInfo** – работа с каталогами, выполняют операции при помощи статических методов, при помощи экземплярных методов**File(статический) и FileInfo** – работа с файлами, выполняют операции при помощи статических методов, при помощи экземплярных методов**ZipFile –** предоставляет дополнительные возможности для создания архивов. Он позволяет создавать архив из каталогов.

**FileSystemWatcher –** позволяет производить мониторинг активности выбранного каталога, ожидает уведомления файловой системы об изменениях и инициирует события при изменениях каталога или файла в каталоге.

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

**using (получение-ресурса)**Здесь получение-ресурса означает один из вариантов.1)Объявление и инициализацию локальной переменной (или списка переменных). Тип переменной должен реализовывать IDisposable. Такая переменная в блоке using доступна только для чтения.2)Выражение, значение которого имеет тип, реализующий IDisposable.using (ClassWithDispose x = new ClassWithDispose()){ x.DoSomething(); // компилятор C# поместит сюда вызов x.Dispose()}**Чтение и запись файлов. Потоковые классы:**

1)типы для представления потоков данных2)адаптеры потоков**Поток данных** – это абстрактное представление данных в виде последовательности байт.1)ассоциируется с неким физическим хранилищем (файлами на диске, памятью, сетью)2)декорирует другой поток, преобразуя данные тем или иным образом

Адаптеры потоков служат оболочкой потока, преобразуя информацию определённого формата в набор байт (сами адаптеры потоками не являются). **FileStream** – класс для работы с файлами, как с потоками (System.IO). **MemoryStream** – класс для представления потока в памяти (System.IO). **NetworkStream** – работа с сокетами, как с потоками (System.Net.Sockets). **PipeStream** – абстрактный класс из пространства имён System.IO.Pipes, базовый для классов-потоков, которые позволяют передавать данные между процессами операционной системы. **StreamReader:**

считывать весь текст или отдельные строки из текстового файла.StreamReader(string path): через параметр path передается путь к считываемому файлуStreamReader(string path, System.Text.Encoding encoding): параметр encoding задает кодировку для чтения файла**StreamWriter:**

Для записи в текстовый файлStreamWriter(writePath, false, System.Text.Encoding.Default).

**49. Классы адаптеры потоков.**

**BinaryReader и BinaryWriter** – классы для ввода и вывода примитивных типов в двоичном формате. **StreamReader и StreamWriter** – классы для ввода и вывода информации в строковом представлении. **XmlReader и XmlWriter** – абстрактные классы для ввода/вывода XML.

**50. Сериализация. Форматы сериализации.**

**Сериализация** - процесс преобразования объектов или связанных объектов в поток байт (диск, память, сеть)**Десериализация** - получение из потока байт сохраненного объектаОбъекты сериализуемых типов можно сохранить в поток в различных форматах, для каждого формата предусмотрен свой класс**Форматы:**

1)бинарный -BinaryFormatter2)SOAP -SoapFormatter3)xml - XmlSerializer4)JSON - DataContractJsonSerializer

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

**Контракт данных** – это тип (класс или структура), объект которого описывает информационный фрагмент (открытые поля и свойства) - один из механизмов сериализацииОсновным форматом хранения контрактов данных является XMLв качестве контракта данных используется обычный класс, информационный фрагмент образуют открытые поля и свойства классаВидимость не имеет значения**Интерфейс ISerializable:**

позволяет выполнить любые действия, связанные с формированием данных для сохранения (свой сериализатор)

public interface ISerializable { void GetObjectData(SerializationInfo info, StreamingContext context);}

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

**Атрибуты контрактов данных:**

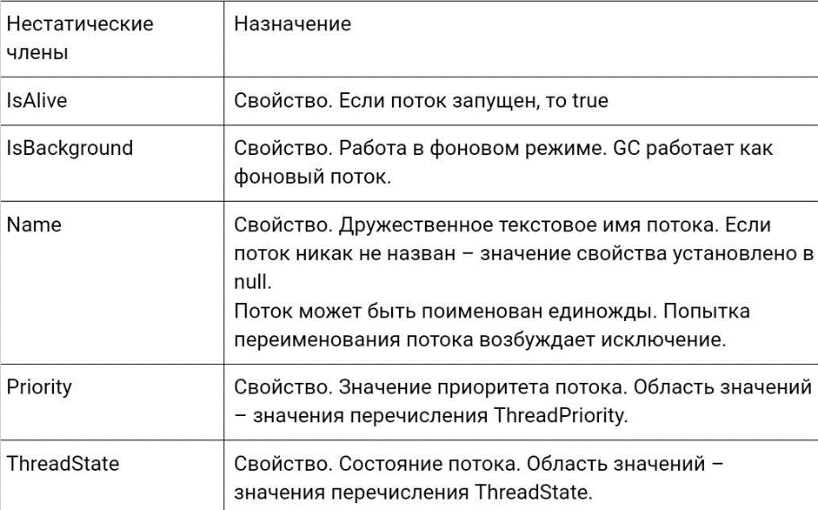
[CollectionDataContract] - Для коллекций[DataContract]свойства Name Namespace для указания имени и пространства имён корневого XML-элемент[DataMember] свойство Name, Order (порядок записи элементов контракта),IsRequired (обязательный элемент для записи), EmitDefaultValue (нужна ли запись значения по умолчанию для элемента)Если контракт будет десериализоваться в объекты потомков своего типа, эти типы должны быть упомянуты при помощи атрибута [KnownType]Если контракт является коллекцией объектов (как класс Group), он маркируется атрибутом [CollectionDataContract].Кроме этого, для методов контракта данных применимы атрибуты [OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized].**Атрибуты:**

атрибут (attribute) - специальный инструмент,который позволяет встраивать в сборку дополнительные метаданные. дополнительная информация, сохраняемая в метаданных о сборке, модуле, типе, элементах типа, параметров методаНаследуются от System.Attribute**Создание собственного атрибута:**

На основе класса Требования к классу:Класс должен прямо или косвенно наследоваться от класса Attribute Тип открытых полей, свойств и параметров конструктора класса: числовые типы (кроме decimal), bool, char, string, object, System.Type, перечисления; одномерные массивыИмя класса должно заканчиваться суффиксом Attribute (необязательно). public class BSTUAttribute : System.Attribute { public string Name { get; private set; } public int Version { get; set; } public BSTUAttribute(string name) { Name = name; } }

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

Приложению в операционной системе соответствует – **процесс** (концепция уровня ОС). Процесс выделяет для приложения изолированное адресное пространство и поддерживает один или несколько потоков выполнения. 1) для каждого загружаемого в память файла \*.ехе в операционной системе создается отдельный изолированный процесс, который используется на протяжении всего времени его существования2) выход из строя одного процесса никак не сказывается на работе других процессов3) доступ напрямую к данным в одном процессе из другого процесса невозможен (API - распределенных вычислений Windows Communication Foundation)4) каждый процесс Windows получает уникальный идентификатор процесса (Process ID — PID) 5) может независимо загружаться и выгружаться операционной системой (в том числе программно) В .NET исполняемые файлы не обслуживаются прямо внутри процесса Windows. ОНИ обслуживаются в отдельном логическом разделе внутри процесса, который называется **доменом приложения** (Application Domain — AppDomain)В процессе может содержаться несколько доменов приложений1) существуют внутри процессов 2) содержат загруженные сборки 3) процесс запускает при старте домен по умолчанию (AppDomain.CurrentDomain)4) домены могут создаваться и уничтожаться в ходе работы в рамках процесса (менее затраты по сравн. с процессами)5) обеспечивают уровень изоляции кода**Поток** - используемый внутри процесса путь выполнения1)CLR поддерживает многопоточность опирается на многопот . ОС2)В каждом процессе Windows содержится первоначальный "поток", который является входной точкой для приложения (метод Main())поток, который создается первым во входной точке процесса, называется главным потоком (primary thread). 3)Главный поток создается автоматически4)Процессы, в которых содержится единственный главный поток выполнения, изначально являются безопасными потоками (thread safe),два типа потоков:1)основной2)фоновыйесли первым завершится основной поток, то фоновые потоки в его процессе будут также принудительно остановленыесли же первым завершится фоновый поток, то это не повлияет на остановку основного потока — тот будет продолжать функционировать до тех пор, пока не выполнит всю работу и самостоятельно не остановится

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

**перечисление ThreadPriority:**1)Lowest2)BelowNormal3)Normal (по умолчанию)4)AboveNormal5)Highest

public Thread(ThreadStart start);public Thread(ParameterizedThreadStart start);public Thread(ThreadStart start, int maxStackSize);

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

1)координация действий для получения предсказуемого результата

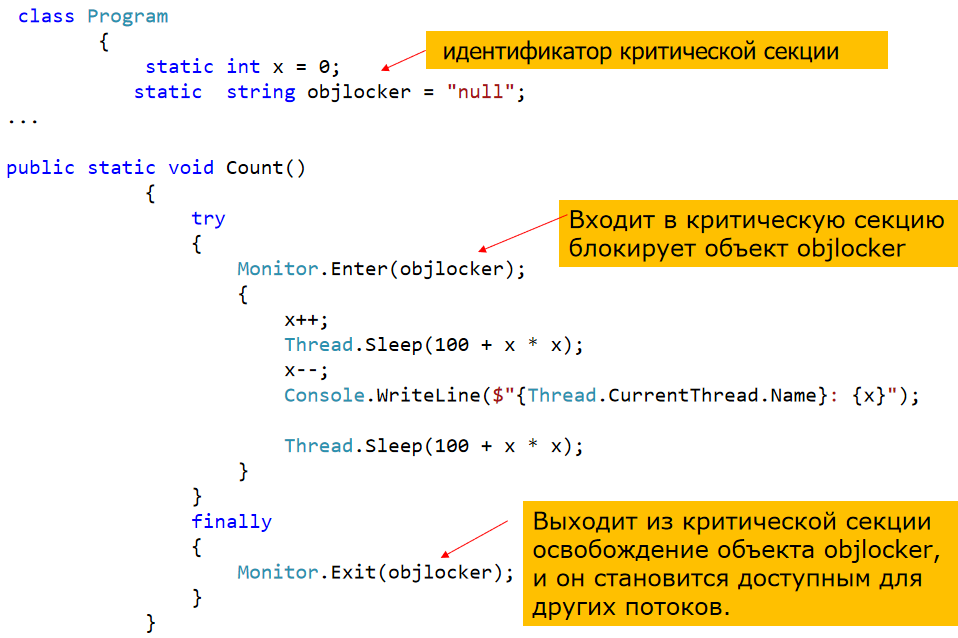
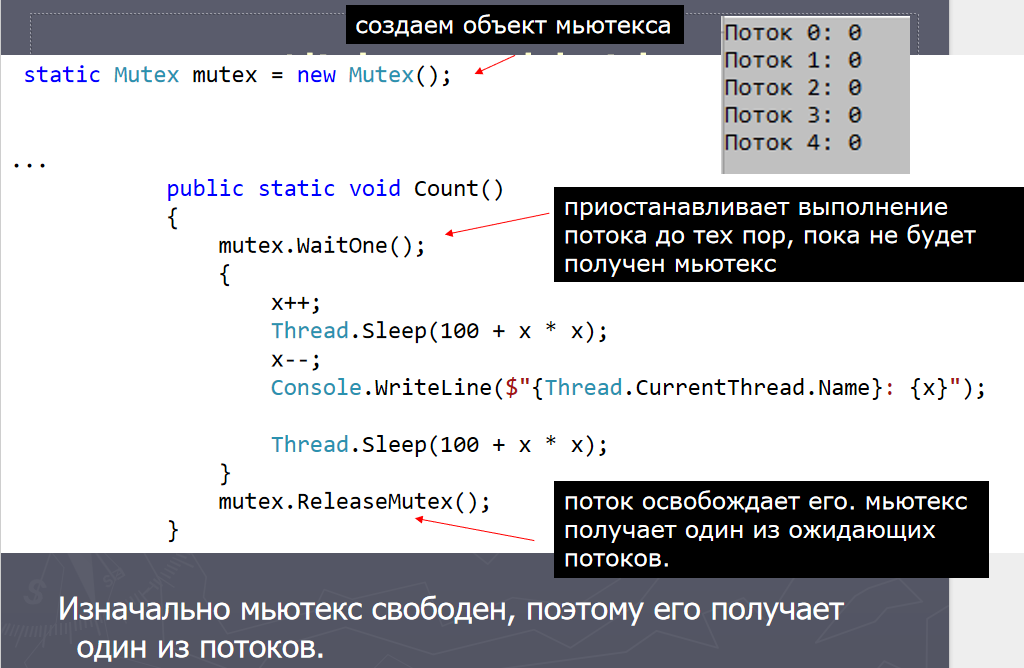
2)В потоках используются разделяемые ресурсы, общие для всей программы**Способы синхронизации:**

1)Монитор (Monitor)2)AutoResetEvent3)Мьютекс (Mutex)4)Семафор (Semaphore)**Оператор Lock:**

определяет блок кода, внутри которого весь код блокируется и становится недоступным для других потоков до завершения работы текущего потока.Остальные потоки помещаются в очередь ожидания и ждут, пока текущий поток не освободит данный блок кода. Необходимо:

1)Как можно быстрее освобождать блокировку2)Избегать взаимоблокировок3)Блокировать только ссылочную переменную 4)Экземпляр объекта должен быть один и тот же для всех потоков

**Monitor:**

механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к неразделяемым ресурсам.1)Monitor.Enter() - вход в критическую секцию, увеличение блокировок на 12)Monitor.Exit() – выход из секции (-1 блок-ка)3)Вход и выход должны выполняться в одном и том же потоке. 4)Аргументами методов является объект-идентификатор критической секции.

**Мьютекс:**

1)System.Threading.Mutex 2)позволяет организовать критическую секцию для нескольких процессов3)WaitOne() - входа в критическую секцию, 4)ReleaseMutex() – для выхода из неё (выход может быть произведён только в том же потоке выполнения, что и вход). **Семафор:**

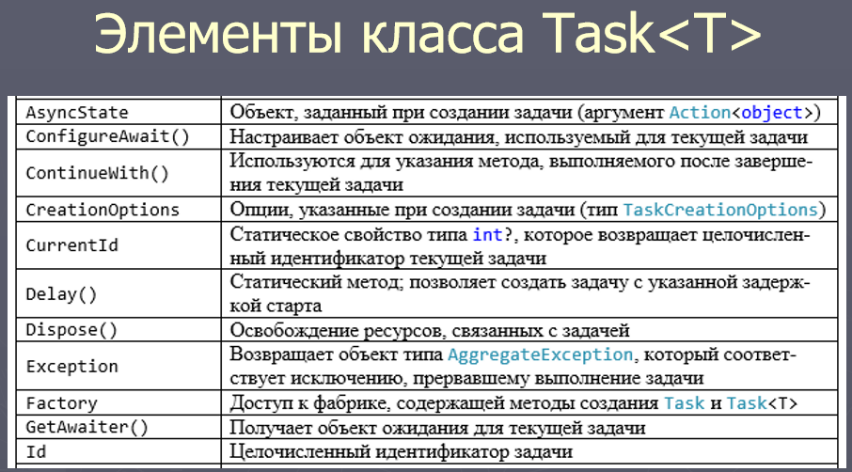
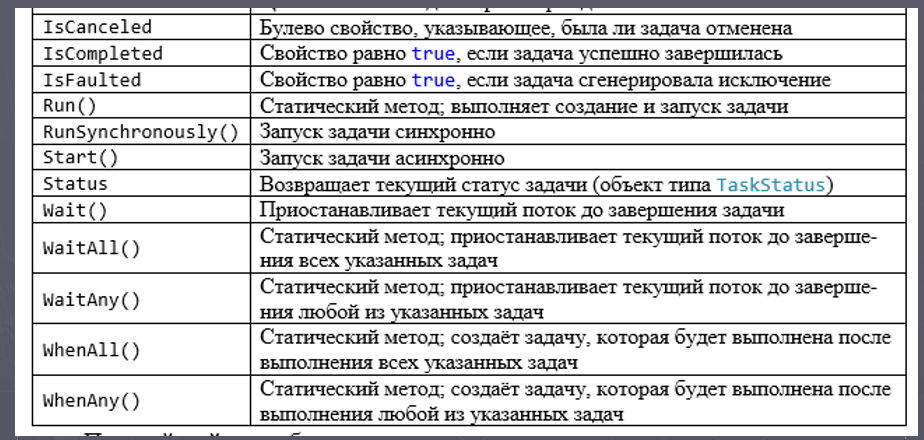
1)объект синхронизации, позволяющий войти в заданный участок кода не более чем N потокам (N – ёмкость семафора)2)получение и снятие блокировки в случае семафора может выполняться из разных потоках3)классы System.Threading.Semaphore (между процессами) и SemaphoreSlim (в рамках одного процесса)4)Wait() - получение блокировки, 5)Release() – снятие блокировкиКонструкторы:

1)Semaphore (int initialCount, int maximumCount): initialCount задает начальное количество потоков, maximumCount - максимальное количество потоков, которые имеют доступ к общим ресурсам2)Semaphore (int initialCount, int maximumCount, string? name): в дополнение задает имя семафора3)Semaphore (int initialCount, int maximumCount, string? name, out bool createdNew): createdNew при значении true указывает, что новый семафор был успешно создан. Если этот параметр равен false, то семафор с указанным именем уже существует

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

**TPL:**

позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах (для создания многопоточных приложений)**Задача (task)** – абстракция более высокого уровня чем потокusing System.Threading.Tasks**Класс Task:**

описывает отдельную продолжительную операцию, которая запускается асинхронно в одном из потоков из пула потоков (можно запускать синхронно в текущем потоке) – подобна потокам, но абстракция более высокого уровняПредставлена .Net 4.0Среда WinRT

**Состояния задачи:**

1)Canceled: задача отменена2)Created: задача создана, но еще не запущена3)Faulted: в процессе работы задачи произошло исключение4)RanToCompletion: задача успешно завершена5)Running: задача запущена, но еще не завершена6)WaitingForActivation: задача ожидает активации и постановки в график выполнения7)WaitingForChildrenToComplete: задача завершена и теперь ожидает завершения прикрепленных к ней дочерних задач8)WaitingToRun: задача поставлена в график выполнения, но еще не начала свое выполнение

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

1)Первый способ создание объекта Task и вызов у него метода Start:

объект Task принимает делегат Action, то есть мы можем передать любое действие, которое соответствует данному делегату2) Второй способ заключается в использовании статического метода Task.Factory.StartNew()

принимает делегат Action, который указывает, какое действие будет выполняться. При этом этот метод сразу же запускает задачу.

3) Третий способ определения и запуска задач представляет использование статического метода Task.Run():

может принимать делегат Action - выполняемое действие и возвращает объект Task.**Wait:** блокирует вызывающий поток, в котором запущена задача, пока эта задача не завершит свое выполнение**Task.WaitAll(tasks):** Если необходимо завершить выполнение программы или вообще выполнять некоторый код лишь после того, как все задачи из массива завершатся**Task.WaitAny(tasks)** - ждет, пока завершится хотя бы одна из массива задач.**Возврат результата:**

Task<TResult> - описывает задачу, возвращающую значение типа Tresultпринимают аргументы типа Func<TResult> Func<object, TResult> (опционально – аргументы типа CancellationToken и TaskCreationOptions)Структура **CancellationToken** - токен отменыCancellationTokenSource tokenSource = new CancellationTokenSource();// используем токен в двух задачах new Task(method, tokenSource.Token).Start();new Task(method, tokenSource.Token).Start(); // отменяем задачи tokenSource.Cancel(); **Продолжения:**

сообщает задаче, что после её завершения она должна продолжить делать что-то другое

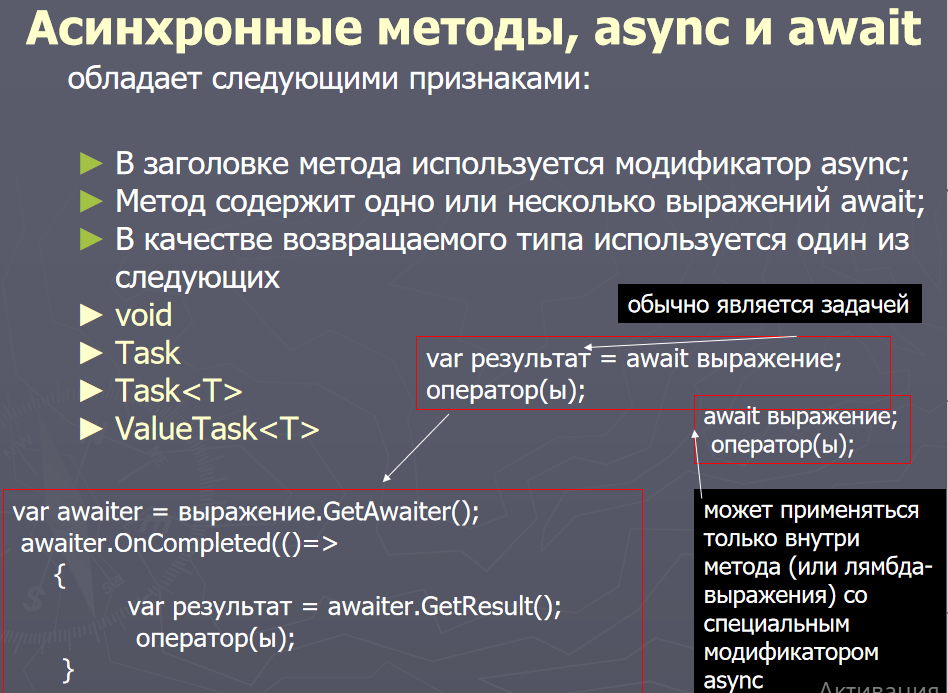
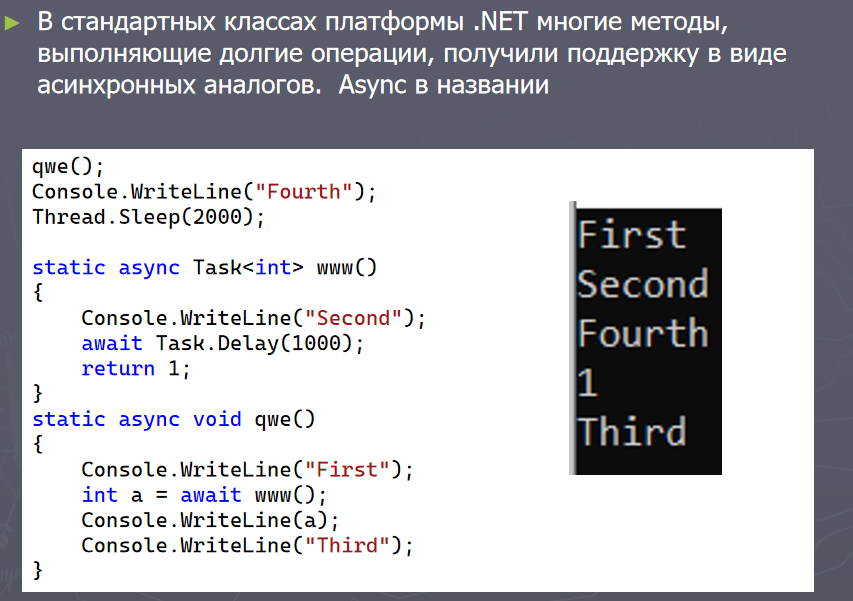
Task task6 = Task.Run(() => Console.Write("Doing.."));Task task7 = task6.ContinueWith(t =>Console.Write("continuation"));

Task task8 = Task.Run(() => Console.Write("One...."));Task task9 = Task.Run(() => Console.Write("Two..."));Task continuation = Task.WhenAll(task8, task9).ContinueWith(t => Console.Write("Three...."));

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

System.Threading.Tasks.Parallel позволяет распараллеливать циклы и последовательность блоков кодаFor(), ForEach(), Invoke() – шаблоны (на задачах, поддерж искл. и токен отмены) являются параллельными аналогами циклов for и foreach**Parallel.For(int, int, Action<int>)** – указание начального и конечного значения счётчика (типа int или long) и тела цикла в виде объекта делегатаParallelLoopResult ForEach<TSource>(IEnumerable<TSource> source,Action<TSource> body)**Parallel.Invoke()** – позволяет распараллелить исполнение блоков операторов – набор задач, которые выполняются в одном потоке

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



await приостанавливает выполнение текущего метода, при этом последовательность выполнения инструкций во внешнем контексте продолжается.

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

Проектирование отношений в объектно-ориентированном программировании (ООП) включает в себя использование различных типов связей между классами. Основные типы отношений — это агрегация, композиция и ассоциация. Давайте рассмотрим каждый из этих типов более подробно.

**Ассоциация** — это общее отношение между двумя классами, когда один класс использует или ссылается на другой. Это отношение может быть однонаправленным или двунаправленным.

1)Односторонняя ассоциация: один класс знает о другом, но не наоборот.

2)Двусторонняя ассоциация: оба класса знают друг о друге.

**Агрегация** — это специальный вид ассоциации, который указывает на "часть-целое" отношение, но без жесткой зависимости между частями. В этом случае, при уничтожении целого, части могут продолжать существовать.

**Композиция** — это более строгий вид агрегации, который также указывает на "часть-целое" отношение, но в этом случае части не могут существовать независимо от целого. При уничтожении целого, части также уничтожаются.

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

1) Spaggeticode(Спагетти-код — слабо структурированная и плохо спроектированная система, запутанная и очень сложная для понимания.извилистый и очень запутанныйзапутанный код)

2) Golden hammer (золотой молоток) – одно решение для всех задач, уверенность в полной универсальности кода3) Code duplication (copy paste) 4) Magic numbers(константы, используемые в коде, но которые не несут никакого смысла без соответствующего комментария)

5) Hard Code (жесткий код) - фиксация в коде различных данных об окружении.6) Soft code(много абстракции, много настроек (сложно и непрозрачно), параноидальная боязнь хард-кода. Этот анти-паттерн является вторым концом палки о хард-коде и поэтому тоже является опасным.7) Academicals complexity(заумность решения, ненужная сложность может быть внесена в решение любой задачи).

8) Boad anchor (лодочный якорь, сохранение неиспользуемых частей системы, которые остались после оптимизации или рефакторинга).

9) Reinvention the wheal(изобретение колеса/велосипеда, разработчик реализует собственное решение для задачи, для которой уже существуют решения, которое может быть лучшие, чем придуманное).

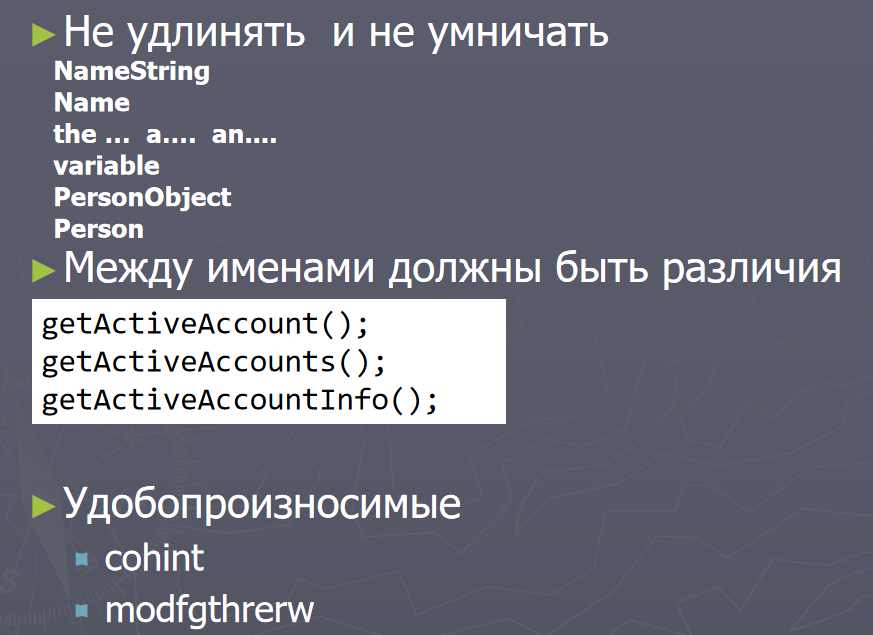
10) Blind faith (слепая вера)(строка вместо числа, проверки внешних параметров) : недостаточная проверка корректности входных данных, отсутствие тестирования при разработки кода и исправлении ошибок.11) God Object (божественный объект)(один класс)12) Programming by permutation (Программирование методом подбора)

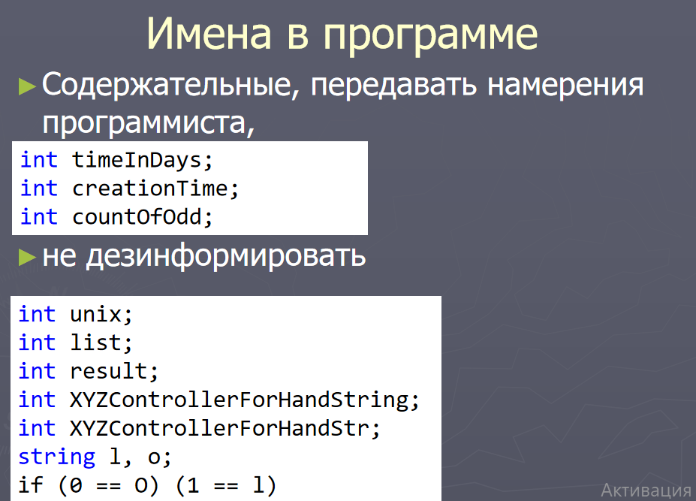
**Рефакторинг** – (англ. refactoring) или реорганизация кода — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы.

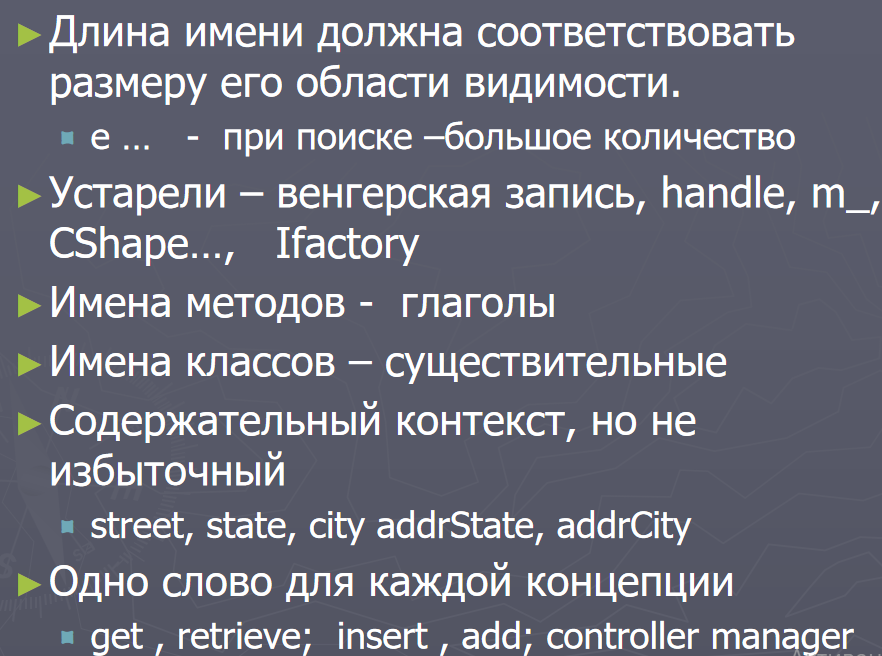
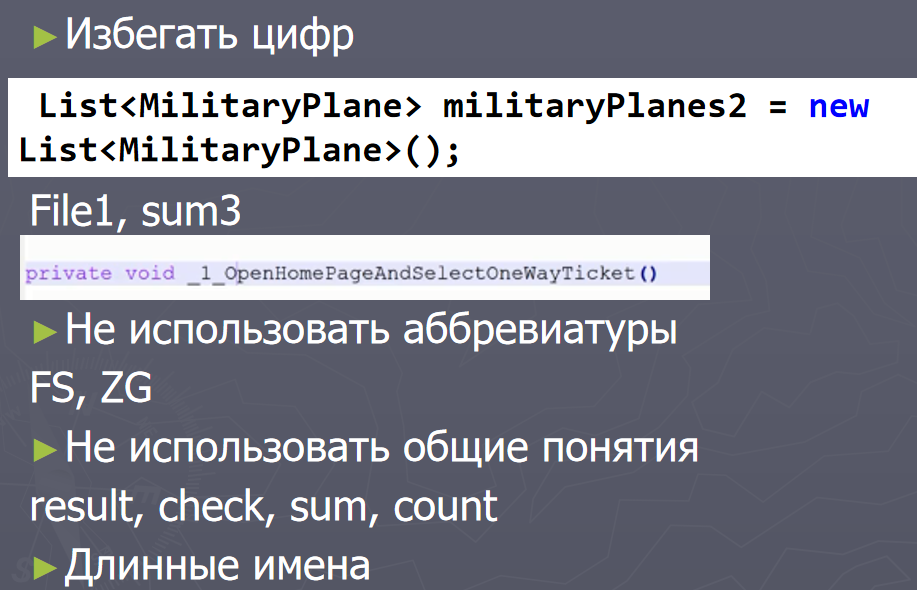
**Методы рефакторинга:**

1)Изменение сигнатуры метода2)Инкапсуляция поля3)Выделение класса4)Выделение интерфейса5)Выделение метода6)Встраивание (Inline)7)Введение фабрики

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

1)логика прямолинейная2)зависимости — минимальные3)стратегия обработки ошибок 4)производительность — близка к оптимальной5)читабельный6)компактный7)решает одну задачу8)невозможно улучшить





**Функции:**

1)Компактность2)Удобно при чтении3)Блоки и отступы4)ФУНКЦИЯ ДОЛЖНА ВЫПОЛНЯТЬ ТОЛЬКО ОДНУ ОПЕРАЦИЮ – Определяется уровнем абстракции 5) Чем меньше аргументов у функции, тем лучше (0, 1, 2, >3 – плохо, можно заменять объектами )6)Возвращать значения через аргументы- плохо7)Аргументы –флаги плохо (сл. Функция выполняет более одной операции)

8)Отделение команд от запроса9)Побочные эффекты функций: неожиданные изменения в переменных класса, вредные привязки и зависимости10) Используйте исключения вместо возвращения кодов ошибок11)Исключайте дублирование

12)Имя должно соответствовать действию и возвращаемому значению13)Используйте конструкцию глагол-объект для именования методов**Форматирование:**

1) понятное название файла2) предпочтительней небольшой размер файла3) вертикальное сжатие4) вертикальное разделение концепций5) тесно связанные друг с другом концепции , должны находиться поблизости друг от друга по вертикали Переменные следует объявлять как можно ближе к месту использования. +зависимые функции Переменные экземпляров должны объявляться в начале класса. вызываемая функция должна располагаться ниже вызывающей функции (сверху-вниз).6)Горизонтальное форматирование7)Длина строки <120 (80) символов (экран)8)Горизонтальное разделение и сжатие: знаки разделяют пробелами, имена функции не отделяются от аргументов9)Отступы: размер отступа соответствует позиции в иерархии

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

1)логика прямолинейная2)зависимости — минимальные3)стратегия обработки ошибок 4)производительность — близка к оптимальной5)читабельный6)компактный7)решает одну задачу8)невозможно улучшить**Объекты:**

Закон Деметры1)модуль не должен знать внутреннее устройство тех объектов, с которыми он работает2)метод f класса C должен ограничиваться вызовом методов следующих объектов: C; объекты, созданные f; объекты, переданные f в качестве аргумента; объекты, хранящиеся в переменной экземпляра C**Класс:**

Последовательность описания 1)открытые статические константы2) приватные статические переменные3) приватные переменные 4) защищённые5) открытых обычно нет6) открытые функции7) приватные функции (м.б. около открытых)8) классы должны быть компактными и еще компактнее9) Имя файла должно соответствовать имени классу

10)Имя – существительное11)Не называть аббревиатурами12)Не давать не понятных названий 13)Не использовать цифры