**Объектно Ориентированное Программирование**

*Полезные ссылки*

***Класс -*** *шаблон для создания объектов*

### ***Абстрактные*** *классы: От обычных классов они отличаются тем, что нельзя создать объект такого класса. Нужен что бы от него могли наследоваться потомки.*

***Объект -*** *некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства и операции над ними.*

***Метод -*** *это функция или процедура, принадлежащая какому-то классу или объекту.*

***Поле(атрибут) -*** *переменная.*

**Основы ООП**

***Абстракция***

*Моделирование требуемых атрибутов и взаимодействий сущностей в виде классов для определения абстрактного представления системы.*

***Инкапсуляция***

*это контроль доступа к полям и методам объекта. Под контролем доступа подразумевается не только можно/не можно, но и различные валидации, подгрузки, вычисления и прочее динамическое поведение. Для этого существуют модификаторы доступа.*

***Полиморфизм***

*свойство системы, позволяющее иметь множество реализаций одного интерфейса.*

***Пример***

Положим, у нас есть три трансформера: Оптимус, Мегатрон и Олег. Трансформеры боевые, стало быть обладают методом attack(). Игрок, нажимая у себя на джойстике кнопку «воевать», сообщает игре, чтобы та вызвала метод attack() у трансформера, за которого играет игрок. Но поскольку трансформеры разные, а игра интересная, каждый из них будет атаковать каким-то своим способом. Скажем, Оптимус — объект класса Автобот, а Автоботы снабжаются пушками с плутониевыми боеголовками (да не прогневаются фанаты трансформеров). Мегатрон — Десептикон, и стреляет из плазменной пушки. Олег — басист, и он обзывается. А в чем польза?  
  
Польза полиморфизма в данном примере заключается в том, что код игры ничего не знает о реализации его просьбы, кто как должен атаковать, его задача просто вызвать метод attack(), сигнатура которого одинакова для всех классов персонажей. Это позволяет добавлять новые классы персонажей, или менять методы существующих, не меняя код игры. Это удобно

***Наследование***

*это механизм системы, который позволяет, как бы парадоксально это не звучало, наследовать одними классами свойства и поведение других классов для дальнейшего расширения или модификации*

**Композиция**

жизненный цикл дочернего объекта совпадает с жизненным циклом родительского

**Доп.**

**сущность А является сущностью Б**? Если да, то скорее всего, тут подойдёт наследование. Если же **сущность А является частью сущности Б**, то наш выбор — композиция

наследование имеет статическую природу и устанавливает отношения классов только на этапе интерпретации/компиляции. Композиция же, как мы видели в примерах, позволяет менять отношение сущностей на лету прямо в рантайме.

**Агрегация**

жизненный цикл дочернего объекта не зависит от жизненного цикла родительского, и может использоваться другими объектами

**Перегрузка**

Если в классе потомке переопределить уже существующий метод в классе родителе, то сработает перегрузка. Это позволяет не дополнять поведение родительского класса, а модифицировать. В момент вызова метода или обращения к полю объекта, поиск атрибута происходит от потомка к самому корню — родителю

**Аксессоры (геттеры и сеттеры)**

Геттеры и сеттеры — это методы, задача которых контролировать доступ к полям. Геттер считывает и возвращают значение поля, а сеттер — наоборот, принимает в качестве аргумента значение и записывает в поле.

**Интерфейсы (дописать)**

*Интерфейс представляет ссылочный тип, который может определять некоторый функционал - набор методов и свойств без реализации. Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы.*

Задача интерфейса — снизить уровень зависимости сущностей друг от друга, добавив больше абстракции.

Определять следующие сущности: Методы, Свойства, Индексаторы, События, Статические поля и константы (начиная с версии C# 8.0), Абстрактный класс.

По умолчанию доступ public.

**утиной типизацией**: *если что-то крякает как утка, плавает как утка, и выглядит как утка, то, скорее всего — это утка*.

**Модификаторы доступа**

**private**: закрытый или приватный компонент класса или структуры. Приватный компонент доступен только в рамках своего класса или структуры.

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

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

**internal**: компоненты класса или структуры доступен из любого места кода в той же сборке, однако он недоступен для других программ и сборок.

**protected internal**: совмещает функционал двух модификаторов protected и internal. Такой компонент класса доступен из любого места в текущей сборке и из производных классов, которые могут располагаться в других сборках.

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

**System.Object**

Поддерживает все классы в иерархии классов .NET и предоставляет низкоуровневые службы производным классам. Это окончательный базовый класс всех классов .NET; это корень иерархии типов.

**Структура(Struct)**

Тип структуры представляет собой [тип значения](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/value-types), который может инкапсулировать данные и связанные функции. Для определения типа структуры используется ключевое слово **struct**.

**Перечисление(enum)**

это [тип значения](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/value-types), определённый набором именованных констант применяемого [целочисленного типа](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/integral-numeric-types). Чтобы определить тип перечисления, используйте ключевое слово **enum** и укажите имена enum.

**Code convention**

**Виртуальные методы**

Ключевое слово virtual используется для изменения объявлений методов, свойств, индексаторов и событий и разрешения их переопределения в производном классе.

**Класс типа sealed**

Запретить переопределение методов и свойств: модификатором **sealed**

**Одноимённые методы - перегрузка**

**Ссылки на абстрактные классы и интерфейсы**

**Операторы is или as**

**Переопределение(override)**

Модификатор **override**. Используется для переопределения метода.

**Перегрузка**

**Класс-Контейнер**

**Управляющий класс-Контроллер**

**Ключевое слово Operator**

**Конструктор**

**деструктор (destructor)**

**this**

**Readonly**

**анонимный тип**

**Static**

статические классы

статическое поле

статических конструкторов

Статическая функция(статический метод)

**Базовый и производный класс**

**.NET**

.NET Framework приложение всегда работает с компонентами с которыми компилировалось и тестировалось приложение.

**Спецификация CLI**

**CLI (Common Language Infrastructure)** – спецификация общеязыковой инфраструктуры. Определяет архитектуру исполнительной системы и набор представляемых сервисов.

Стандарты: ECMA-335 и ISO/IEC 23271

**Microsoft.NET (.NET Framework)** – программная платформа. Содержит следующие основные компоненты:

**CLR (Common Language Runtime**) – обще языковая среда исполнения, виртуальная машина на которой исполняются все приложения, работающие в среде.NET. Реализация CLI VES компанией Microsoft. Компилятор JIT(Just in Time).

**CLR (Common Language Runtime) –** Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки(управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация)

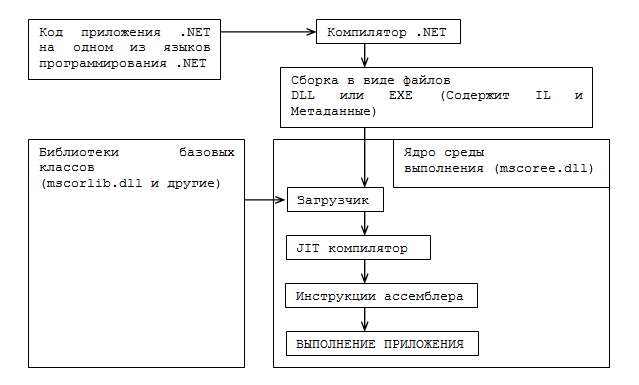
CLR позволяет типы разработанные на одном языке использовать в других языках.

CLR автоматически отслеживает использование ресурсов. Сборщик мусора.

При работе в CLR практически исключена возможность записать (стереть) данные в область памяти, которая для этого не предназначена. Нет озможности передать управление в произвольную точку.

Для обработки ошибок и сбоев в CLR используется только механизм исключений.

**CLR LANGUAGE** : C++/CLI, Visual Basic, F#, Iron Python, Iron Ruby и ассемблер Intermediate Language (IL) Ada, APL, Caml, COBOL, Eiffel, Forth, Fortran, Haskell, Lexico, LISP, LOGO, Lua, Mercury, ML, Mondrian, Oberon, Pascal, Perl, Php, Prolog, RPG, Scheme, Smalltalk

**Структура среды выполнения CLR** 

**MSIL(Microsoft IL)** – реализация CLI CIL компанией Microsoft.

**FCL (Framework Class Library)** – реализация CLI BCL компанией Microsoft. Можно рассматривать, как API CRL.

**FCL (.NET Framework Class Library)** – соответствующая CLS спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений)

**CLS (Common Language Specification)** – обще языковая спецификация, предназначенная для разработчиков компиляторов. CTS (Common Type

Systems)- спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Microsoft выпустил несколько компиляторов соответствующих этой спецификации: С++/CLI (C++ с

управляемыми расширениями), С#, VB .NET, JScript.

**IL**

объектно-ориентированный машинный язык не зависящий от процессора ILAsm.exe – ассемблер

ILDasm.exe - дизассемблер IL

**Метаданные**

двоичный набор таблиц данных: типы и их члены портируемые типы и их члены

**Манифест**

набор таблиц метаданных файлы, которые входят в сборку, общедоступные экспортируемые типы, файлы ресурсов или данных

**Сборки**

Абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов.

Дискретная единица многократно используемого кода внутри CLR.

**Типы сборок:**

с нестрогими именами (weakly named assemblies)

со строгими именами (strongly named assemblies).

**Полезные ссылки**

Трансформеры на хабре

<https://habr.com/ru/post/463125/>

Модификаторы доступа matanit

<https://metanit.com/sharp/tutorial/3.2.php>

Интерфейсы

<https://metanit.com/sharp/tutorial/3.9.php>

<https://habr.com/ru/post/343852/>

Виртуальный (virtual)

<https://metanit.com/sharp/tutorial/3.19.php>

https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/virtual

Одноимённые методы - перегрузка

(https://metanit.com/sharp/tutorial/3.5.php)

Ссылки на абстрактные классы и интерфейсы

https://professorweb.ru/my/csharp/charp\_theory/level9/9\_2.php

Перечисление

**<https://metanit.com/sharp/tutorial/2.12.php>**

**https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/enum**

Класс-Контейнер

**<https://learn.microsoft.com/ru-ru/dotnet/api/system.componentmodel.container?view=net-6.0>**

Структуры

<https://metanit.com/sharp/tutorial/2.13.php>

https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/struct

System.Object

<https://metanit.com/sharp/tutorial/3.10.php>

https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-7.0