Карагандинский технический университет имени Абылкаса Сагинова
Кафедра «Кибербезопасность и искусственный интеллект»

СЛАЙД-ЛЕКЦИЯ

Тема: «Введение в C#. Язык C# и платформа .NET»

Дисциплина: Программирование на C#

Специальность: ВТиПО

Автор: Ст.преп. Сницарь Лилия Ринатовна

ПЛАН ЛЕКЦИИ

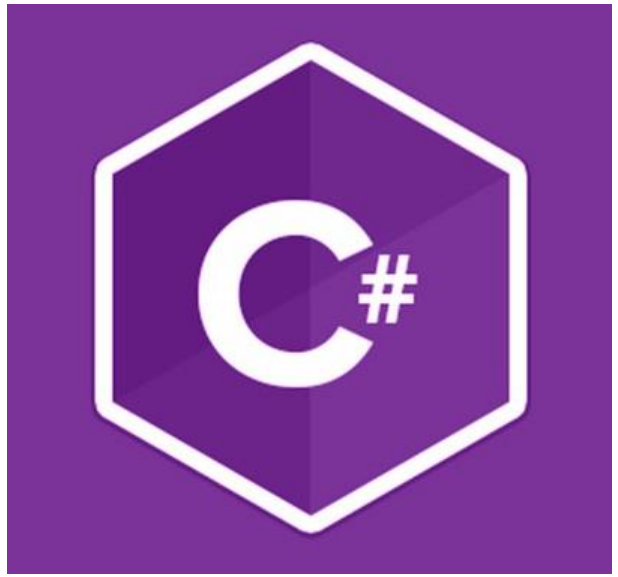
1. Язык C# и платформа .NET.
2. Мотивация изучения.
3. Основные компоненты .NET.
4. CTS, CLS, CLR.
5. Пространства имен.
6. Основы ООП.

1. Язык C# и платформа .NET

C# является объектно-ориентированным языком с Си-подобным синтаксисом.

C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию.

Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.



1. Язык C# и платформа .NET

Ключевые функциональные возможности C#:

- Не нужно никаких указателей;
- Автоматическая сборка мусора (автоматическое управление памятью);
- Формальные синтаксические конструкции для классов, интерфейсов, структур, перечислений и делегатов;
- Возможность перегрузки операторов для пользовательских типов;
- Поддержка анонимных методов, позволяющих предоставлять встраиваемую функцию везде, где требуется использовать тип делегата.

1. Язык C# и платформа .NET

Роль платформы .NET

Язык C# был создан специально для работы с фреймворком .NET. Фреймворк .NET представляет мощную платформу для создания приложений.

Можно выделить следующие ее основные черты:

- Поддержка нескольких языков;
- Кроссплатформенность;
- Мощная библиотека классов;
- Разнообразие технологий;
- Производительность.

1. Язык C# и платформа .NET

Управляемый и неуправляемый код

Нередко приложение, созданное на C#, называют **управляемым кодом** (managed code).

Это значит, что данное приложение создано на основе платформы .NET и поэтому управляется общеязыковой средой CLR, которая загружает приложение и при необходимости очищает память.

В то же время платформа .NET предоставляет возможности для взаимодействия с неуправляемым кодом.

1. Язык C# и платформа .NET

JIT-компиляция

Код на C# компилируется в приложения или сборки с расширениями **exe** или **dll** на языке **CIL** (Common Intermediate Language).

Далее при запуске на выполнение подобного приложения происходит **JIT-компиляция** (Just-In-Time) в машинный код, который затем выполняется.

2. Мотивация изучения

Что такое .NET

.NET – это платформа от Microsoft, которая позволяет создавать программные приложения.

Первый выпуск .NET Framework состоялся в **2002 году**.

Считается, что .NET Framework была создана как альтернатива платформе Java от компании Sun.

Главное отличие состоит в том, что .NET Framework официально рассчитана на работу именно с операционными системами семейства Microsoft Windows.

2. Мотивация изучения

Чем занимаются .NET-разработчики

Языки и платформы созданы для решения конкретных задач и разработки соответствующих программных продуктов.

У .NET тоже есть своя специфика. При этом диапазон продуктов, над созданием которых трудятся .NET-разработчики, очень широкий. Достаточно посмотреть на этот список: в нём перечислены использующие .NET компании из самых разных отраслей – от финансовой и торговой до научной и социальной. В целом, всё разнообразие программных продуктов, которые создаются под .NET, можно сгруппировать следующим образом.

2. Мотивация изучения

Web Development

Чтобы написать web-приложение под .NET, необходимо знать C# и владеть фреймворком ASP.NET MVC.

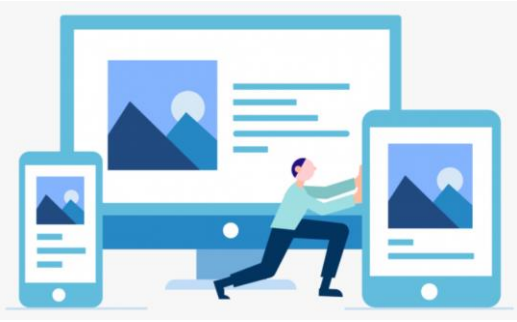
Также нужно понимать, что такое клиент/сервер, как устроен протокол HTTP, REST, JavaScript, различать Frontend и Backend.

И если мы говорим про современную разработку, то важно иметь представление о доменах, хостингах, планах, а ещё – немного об облачных технологиях (MS Azure, Amazon, Google Cloud).

2. Мотивация изучения

Client Applications

Под .NET создаются не только web, но и клиентские приложения – продукты, которые запускаются на персональных компьютерах и мобильных устройствах конечных пользователей.



Например, для трейдеров – NinjaTrader, Tradesignal, для бизнес-аналитиков – Microsoft Power BI, которое визуализирует информацию из любого источника и позволяет проще и быстрее работать с большими данными.

В основном для Desktop Client Applications применяются технологии WPF либо Windows Forms – зная их, вы сможете создавать сложные приложения для стационарных компьютеров пользователей.

2. Мотивация изучения

Game Development

С помощью .NET можно писать игры под Unity.

Под Unity написаны такие известные игры, как Inside, Kerbal Space Program, Endless Legend, Pokemon Go.

Для создания игры достаточно знать язык C# и использовать библиотеки платформ Mono и Unity.

2. Мотивация изучения

Enterprise

Enterprise – это область разработки продуктов для решения проблем бизнеса, а не конечных пользователей.

К таким продуктам можно отнести CRM-системы для отслеживания поведения клиентов, а также системы менеджмента информации или документооборота.

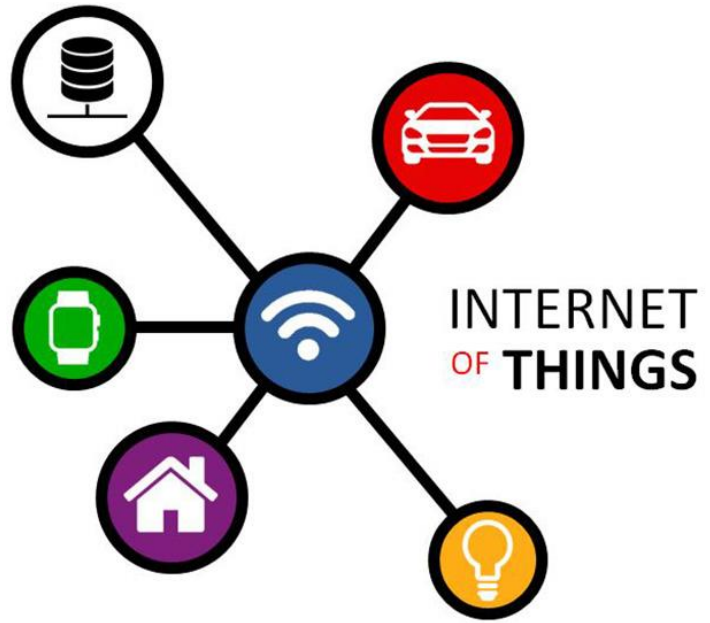
Enterprise-системы являются комбинацией разработки web, desktop и mobile.

Customer Relationship Management

2. Мотивация изучения

Internet of Things (IoT)

Здесь также можно применять .NET, например, если вы используете Raspberry PI с Windows 10 IoT Core и хотите управлять умными чайниками, системами домашнего освещения, беспилотными автомобилями, системами распознавания речи и автоматизированных диалогов на базе готовых фреймворков.



3. Основные компоненты .NET

Три кита: CLR, CTS и CLS

С точки зрения программиста .NET представляет собой исполняющую среду и обширную библиотеку базовых классов.

Уровень исполняющей среды называется **общеязыковой исполняющей средой** (**Common Language Runtime**) или средой **CLR**.

Основная задача **CLR** – автоматическое обнаружение, загрузка и управление типами .NET.

Также среда CLR заботится о ряде низкоуровневых деталей – управление памятью, обработка потоков, выполнение разных проверок, связанных с безопасностью.

3. Основные компоненты .NET

Другой компонент .NET – **общая система типов** (**Common Type System**) или система **CTS**.

Предоставляет полное описание всех возможных типов данных и программных конструкций, которые поддерживаются исполняющей средой, а также способов, как все эти сущности могут взаимодействовать друг с другом.

Нужно понимать, что любая возможность CTS может не поддерживаться в отдельно взятом языке, совместимом с .NET.

3. Основные компоненты .NET

Именно поэтому существует третий компонент – **CLS (Common Language Specification)** или **спецификация CLS**.

В ней описано лишь то подмножество общих типов и программных конструкций, которое способны воспринимать все .NET языки.

3. Основные компоненты .NET

Библиотека базовых классов

Кроме среды CLR и спецификаций CTS и CLS, в составе платформы .NET существует библиотека базовых классов. Она доступна для всех языков, поддерживающих .NET.

В этой библиотеке содержатся определения примитивов (потоки, файловый I/O, системы графической визуализации, механизмы для взаимодействия с разными внешними устройствами), предоставлена поддержка целого ряда служб, которые нужны в большинстве реальных приложений.

3. Основные компоненты .NET

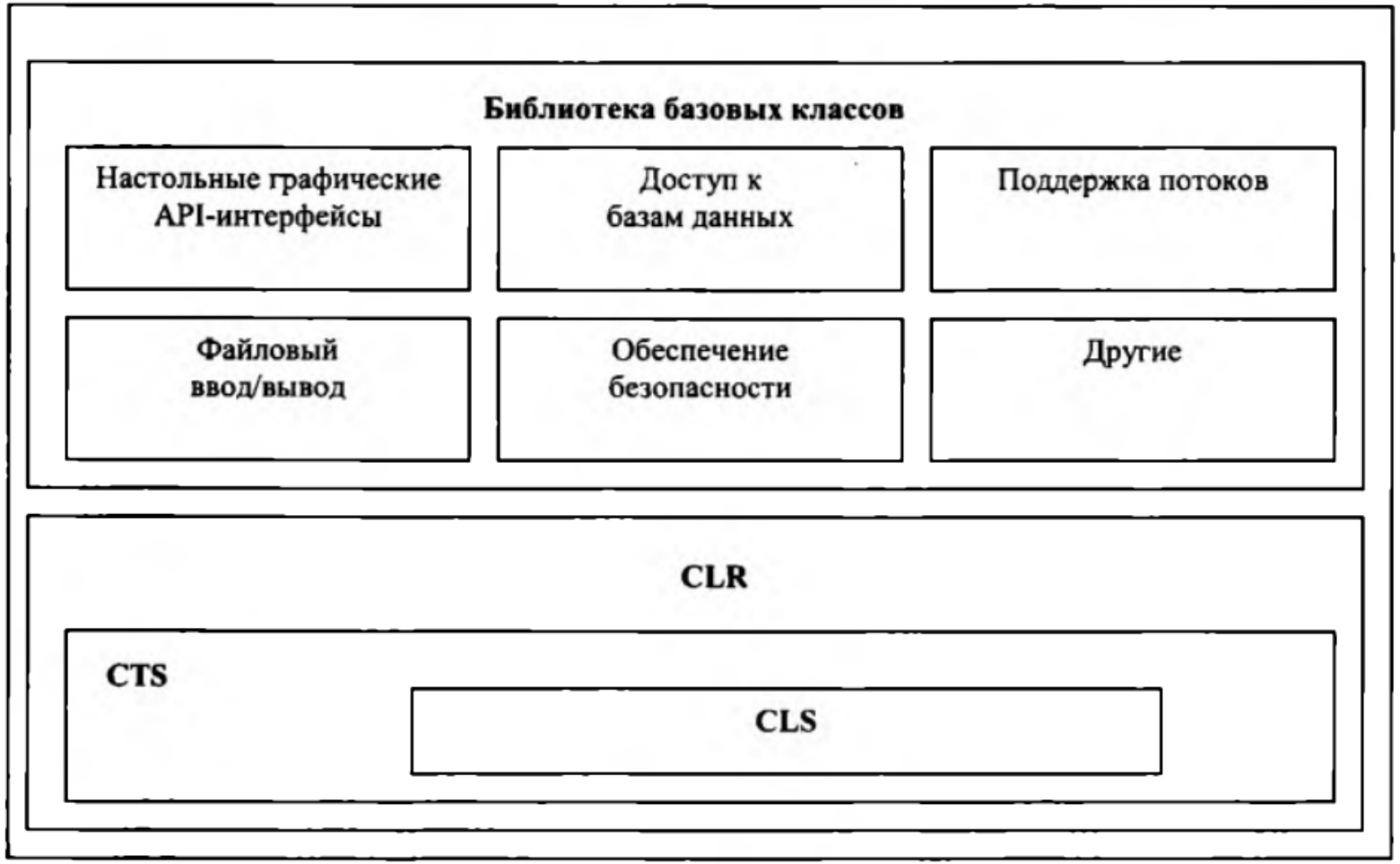


Рис. 1 – Взаимосвязь между компонентами .NET и библиотекой базовых классов

4. CTS, CLS, CLR

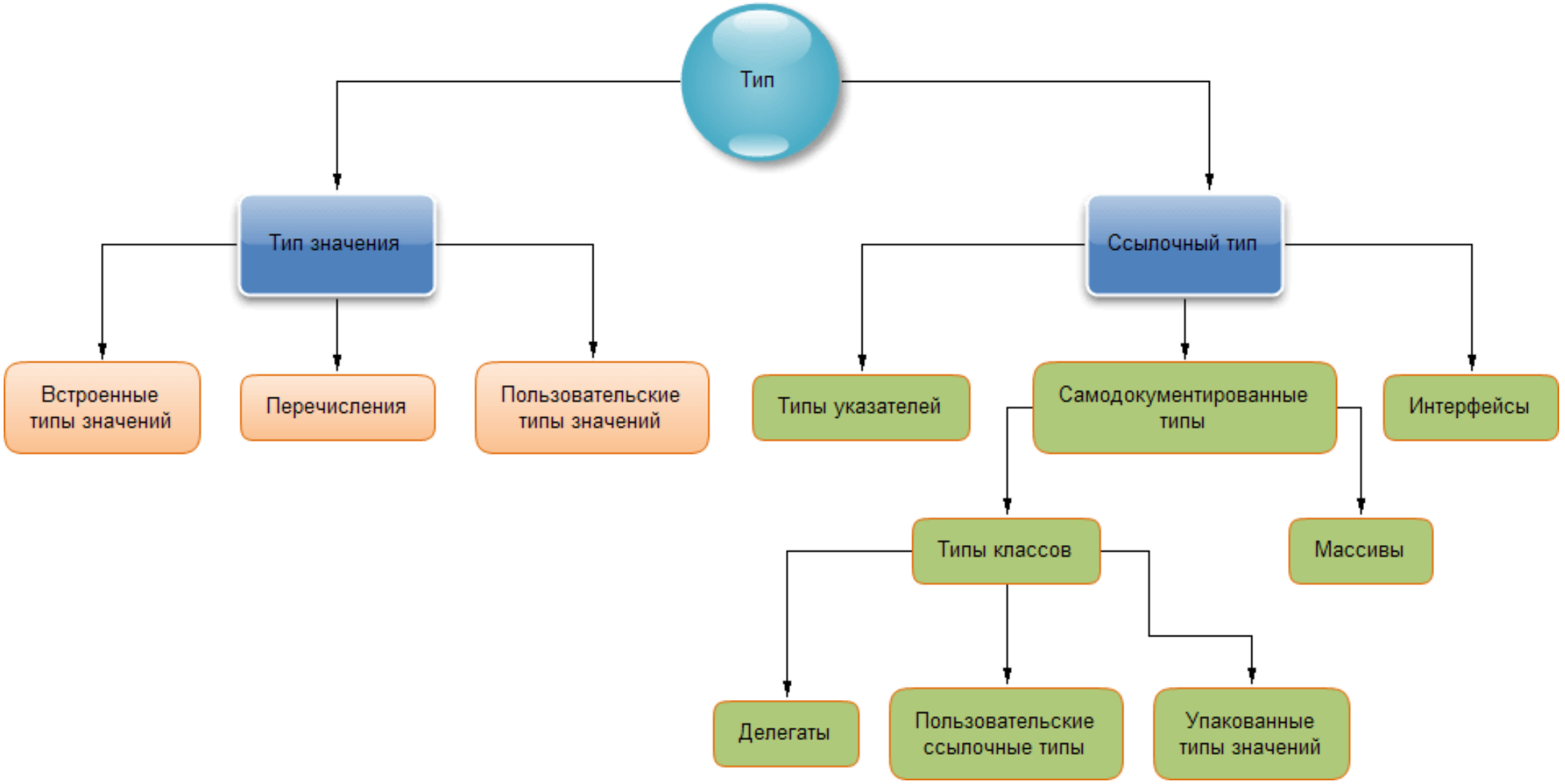
В каждой конкретной сборке может содержаться любое количество самых разных типов.

В мире .NET «тип» – это просто общий термин, который может использоваться для обозначения любого элемента из множества (класс, интерфейс, структура, перечисление, делегат).

Так, в сборке может содержаться один класс, реализующий определенное количество интерфейсов, метод одного из которых может принимать в качестве параметра перечисление, а возвращать – массив.

CTS (общая система типов) представляет собой формальную спецификацию, в которой описано то, как должны быть определены типы для того, чтобы они могли обслуживаться в CLR-среде.

4. CTS, CLS, CLR


Тип
Тип значения
Ссылочный тип
Встроенные типы значений
Перечисления
Пользовательские типы значений
Типы указателей
Самодокументированные типы
Интерфейсы
Типы классов
Массивы
Делегаты
Пользовательские ссылочные типы
Упакованные типы значений

4. CTS, CLS, CLR

Типы классов

В каждом совместимом с .NET языке поддерживается, как минимум, понятие типа класса (class type), которое играет центральную роль в **объектно-ориентированном программировании** (ООП).

Каждый класс может включать в себя любое количество членов (таких как конструкторы, свойства, методы и события) и точек данных (полей).

4. CTS, CLS, CLR

В C# классы объявляются с помощью ключевого слова class.

```
class Summa
{
   public int Sum (int x, int y)
   {
      return x+y;
   }
}
```

4. CTS, CLS, CLR

Таблица 1. Характеристики классов CTS

Характеристика	Описание
Степень видимости	Каждый класс должен настраиваться с атрибутом видимости (visibility). По сути, данный атрибут указывает, должен ли класс быть доступным внешним сборкам или его можно использовать только внутри определенной сборки.
Абстрактные и конкретные классы	Экземпляры абстрактных классов не могут создаваться напрямую и предназначены для определения общих аспектов поведения для произвольных типов. Экземпляры конкретных классов могут создаваться напрямую.
Запечатанные	Запечатанные (sealed) классы не могут выступать в роли базовых для других классов, то есть не поддерживают наследия.
Реализующие интерфейсы	Интерфейс (interface) – это коллекция абстрактных членов, которые обеспечивают возможность взаимодействия между объектом и пользователем этого объекта. CTS позволяет реализовать в классе любое количество интерфейсов.

4. CTS, CLS, CLR

Типы интерфейсов

Интерфейсы представляют собой именованную коллекцию определений абстрактных членов, которые могут поддерживаться в данном классе или структуре. В C# типы интерфейсов определяются с помощью ключевого слова interface.

Сами по себе интерфейсы мало чем полезны. Однако когда они реализуются в классах или структурах уникальным образом, они позволяют получать доступ к дополнительным функциональным возможностям за счет добавления просто ссылки на них в полиморфной форме.

4. CTS, CLS, CLR

Пример объявления интерфейса:

```
public interface ICommandSource
{
    void CommandParameter();
}
```

4. CTS, CLS, CLR

Типы структур

В CTS есть понятие структуры. Простыми словами структура может считаться «облегченной» версией класса. Обычно структуры лучше подходят для моделирования математических данных.

В C# структуры определяются ключевым словом **struct**.

4. CTS, CLS, CLR

```
// Тип структуры в C#
struct Rectangle
{
//  В  структурах  могут  содержаться  поля,  конструкторы  и
//определяться методы
    public string recFill;

    public void MyBackground()
    {
        Console.WriteLine("Фон элемента: "+recFill);
    }
```

4. CTS, CLS, CLR

Типы перечислений

Перечисления (enumeration) – удобная программная конструкция, позволяющая сгруппировать данные в пары «имя-значение». Например:

```
// Тип перечисления C#
public enum CharacterType
{
Wizard = 100,
Fighter = 200,
Thief = 300
}
```

4. CTS, CLS, CLR

Типы делегатов

Делегаты (delegate) – являются .NET-эквивалентом безопасных в отношении типов указателей функций в стиле Си.

Основное отличие заключается в том, что делегат в .NET представляет собой класс, который наследуется от System.MulticastDelegate, а не просто указатель на какой-то конкретный адрес в памяти. Объявить делегат можно с помощью ключевого слова **delegate**:

```
public delegate int AddOp (int x, int y);
```

4. CTS, CLS, CLR

Встроенные типы данных

В CTS также содержится четко определенный набор фундаментальных типов данных.

В каждом отдельно взятом языке для объявления того или иного встроенного типа данных из CTS обычно предусмотрено свое уникальное ключевое слово.

В таблице 2 показано, какие ключевые слова в разных языках соответствуют типам данных в CTS.

4. CTS, CLS, CLR

Таблица 2. Встроенные типы данных в CTS

CTS	C#	C++
System.Byte	byte	unsigned char
System.SByte	SByte	sbyte
System.Int16	short	short
System.Int32	int	int или long
System.Int64	long	_int64
System.Ulnt16	ushort	unsigned short
System.Ulnt32	uint	unsigned int
System.Unit64	ulong	unsigned_int64
System.Single	float	float
System.Double	double	double
System.Object	object	object
System.Char	char	wchar_t
System.String	String	string
System.Decimal	decimal	decimal
System.Boolean	bool	bool

4. CTS, CLS, CLR

При этом в C# можно указать названия типов из CTS:

```
long x = 0;
System.Int64 y = 0;
```

4. CTS, CLS, CLR

CLS (Common Language Specification – общая спецификация для языков программирования) представляет собой набор правил, которые подробно описывают минимальный и полный набор функциональных возможностей, которые должен обязательно поддерживать каждый отдельно взятый .NET-компилятор, чтобы генерировать такой программный код, который мог бы обслуживаться CLR.

CLS можно считать просто подмножеством всех функциональных возможностей, определенных в CTS. В конечном итоге CLS является своего рода набором правил, которых должны придерживаться создатели компиляторов, если они хотят, чтобы их продукты могли без особых проблем функционировать в мире .NET.

4. CTS, CLS, CLR

CLR (Common Language Runtime) – общеязыковая исполняющая среда. CLR можно расценивать как коллекцию внешних служб, которые необходимы для выполнения скомпилированной единицы программного кода.

Например, при использовании платформы MFC для создания нового приложения программист осознает, что приложению понадобится библиотека времени выполнения MFC (то есть mfc40.dll).

5. Пространства имен

Пространства имен (**namespace**) – это способ, благодаря которому .NET избегает конфликтов имен между классами.

Они предназначены для того, чтобы исключить ситуации, когда вы определяете класс, представляющий заказчика, называете его Customer, а после этого кто-то другой делает то же самое (подобный сценарий достаточно распространен).

Пространства имен можно вкладывать друг в друга. Например, большинство базовых классов .NET общего назначения находятся в пространстве имен ***System***.

Базовый класс ***Array*** относится к этому пространству, поэтому его полное имя – ***System.Array***.

5. Пространства имен

Платформа .NET требует, чтобы все имена были объявлены в пределах пространства имен; например, вы можете поместить свой класс ***MyClass*** в пространство имен ***MyCompany***. Тогда полное имя этого класса будет выглядеть как ***MyCompany.MyClass***.

В большинстве ситуаций Microsoft рекомендует применять хотя бы два вложенных пространства имен: первое – *наименование вашей компании*, а второе – *название технологии* или пакета программного обеспечения, к которому относится класс, чтобы это выглядело примерно так: ***MyCompany.SomeNamespace.MyClass***.

В большинстве случаев такой подход защитит классы вашего приложения от возможных конфликтов с именами классов, написанных разработчиками из других компаний.

5. Пространства имен

Для подключения пространства имен в C# используется ключевое слово using, например:

```
using System;
using System.Drewing;
using System.Windows.Forms;
```

5. Пространства имен

Логическая организация кода

Namespaces помогают структурировать проект.

В больших приложениях классы распределяются по смысловым блокам: работа с сетью (System.Net), коллекции (System.Collections), ввод-вывод (System.IO), многопоточность (System.Threading).

5. Пространства имен

Вложенные пространства имен

Можно создавать иерархию:

```
namespace MyCompany.Technology.Product
{
    class MyClass { }
}
```

Полное имя класса будет MyCompany.Technology.Product.MyClass.

5. Пространства имен

Пространства имен уровня файла (File-scoped namespaces)

В C# 10 появилась упрощённая форма объявления:

```
namespace MyCompany.Project;
class MyClass { }
```

Все типы в файле автоматически принадлежат этому пространству имен. Это сокращает вложенность и делает код более читаемым.

5. Пространства имен

Таблица 3. Некоторые пространства имен в .NET

Пространство имен	Описание
System	Содержит много полезных типов, позволяющих работать с математическими вычислениями, генератором случайных чисел, переменными среды и т.д.
System.Collections System.Collections.Generic	Содержит ряд контейнерных типов, а также несколько базовых типов и интерфейсов, позволяющих создавать специальные коллекции
System.Data System.Data.Common System.Data.EntityClient System.Data.SqlClient	Используется для взаимодействия с базами данных через ADO.NET
System.IO System.IO.Compression System.IO.Ports	Здесь содержится много типов, предназначенных для работы с операциями файлового ввода/вывода, сжатия данных, портами
System.Reflection System.Reflection.Emit	Содержит типы, которые поддерживают обнаружение типов во время выполнения, а также динамическое создание типов
System.Runtime System.Runtime.InteropServices	Содержит средства, позволяющие взаимодействовать с неуправляемым кодом, точнее, эти средства находятся в System.Runtime.InteropServiceS
System.Drawing System.Windows.Forms	Содержит типы, применяемые для построения настольных приложений (Windows Forms)

5. Пространства имен

Пространство имен	Описание
System.Windows System.Windows.Controls System.Windows.Shapes	Является корневым для нескольких пространств имен, предоставляющих набор графических инструментов WPF (Windows Presentation Foundation)
System.Linq System.Xml.Linq System.Data.DataSetExtensions	Содержит типы, применяемые при выполнении программирования с использованием API LINQ
System.Web	Позволяет создавать веб-приложения ASP.NET
System.ServiceModel	Позволяет создавать распределенные приложения с помощью API-интерфейса Windows Communication Foundation (WCF)
System.Xml	Здесь содержатся многочисленные типы, которые используются при работе с XML-данными
System.Security	Типы, имеющие дело с разрешениями, криптографией и т.д.
System.Threading System.Threading.Tasks	Средства для создания многопоточных приложений, способных разделить нагрузку среди нескольких процессоров
System.Workflow System.Workflow.Runtime System.Workflow.Activities	Типы для построения поддерживающих рабочие потоки приложений с помощью API-интерфейса Windows Workflow Foundation (WWF)

5. Пространства имен

Практические рекомендации

Используйте минимум два уровня вложенности: компания + проект/технология.

Избегайте слишком длинных пространств имен – они усложняют чтение.

Для больших проектов полезно разделять код по сборкам (assemblies), каждая из которых имеет свои пространства имен.

6. Основы ООП

Все основанные на объектах языки (C#, Java, C++, Smalltalk, Visual Basic и т.п.) должны отвечать трем основным принципам объектно-ориентированного программирования (ООП):

- **Инкапсуляция;**
- **Наследование;**
- **Полиморфизм.**

6. Основы ООП

Инкапсуляция – это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных.

Т.е. инкапсуляция представляет собой способности языка скрывать излишние детали реализации от пользователя объекта.

6. Основы ООП

Например, предположим, что используется класс по имени DatabaseReader, который имеет два главных метода: Open() и Close().

Фиктивный класс DatabaseReader инкапсулирует внутренние детали нахождения, загрузки, манипуляций и закрытия файла данных.

6. Основы ООП

```
public class Person
{
    private string name; // Закрытое поле (private), доступно только внутри класса

    Ссылок: 0
    public string GetName() // Метод для получения имени
    {
        return name;
    }

    Ссылок: 0
    public void SetName(string newName) // Метод для установки имени
    {
        if (!string.IsNullOrEmpty(newName))
        {
            name = newName;
        }
    }
}
```

В этом примере поле name инкапсулировано, и доступ к нему осуществляется через публичные методы GetName и SetName.

Это позволяет контролировать и обеспечивать целостность данных объекта Person.

6. Основы ООП

Основной единицей инкапсуляции в C# является класс, который определяет форму объекта.

Он описывает данные, а также код, который будет ими оперировать. В C# описание класса служит для построения объектов, которые являются экземплярами класса.

Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

6. Основы ООП

Следующий принцип ООП – **наследование** – касается способности языка позволять строить новые определения классов на основе определений существующих классов.

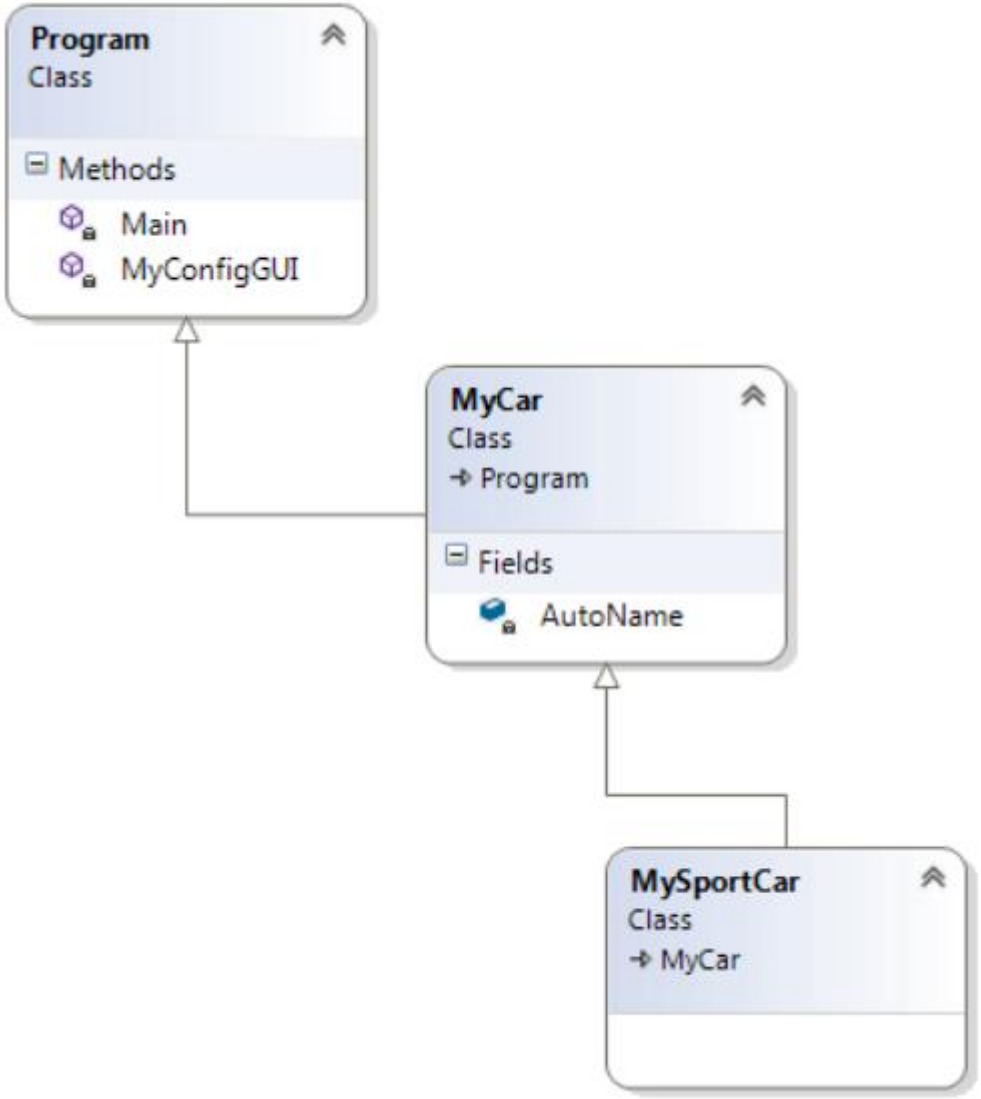
По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом):

6. Основы ООП

Т.е. наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта.

Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации.

Если вдуматься, то большая часть знаний поддается систематизации благодаря иерархической классификации по нисходящей.



6. Основы ООП

Последний принцип ООП – **полиморфизм**. Он обозначает способность языка трактовать связанные объекты в сходной манере.

В частности, этот принцип ООП позволяет базовому классу определять набор членов (формально называемый полиморфным интерфейсом), которые доступны всем наследникам.

Полиморфный интерфейс класса конструируется с использованием любого количества виртуальных или абстрактных членов.

6. Основы ООП

Рассмотрим для примера стек, т.е. область памяти, функционирующую по принципу «последним пришел – первым обслужен».

Допустим, что в программе требуются три разных типа стеков: один – для целых значений, другой – для значений с плавающей точкой, третий – для символьных значений.

В данном примере алгоритм, реализующий все эти стеки, остается неизменным, несмотря на то, что в них сохраняются разнотипные данные.

В языке, не являющемся объектно-ориентированным, для этой цели пришлось бы создать три разных набора стековых подпрограмм с разными именами.

Но благодаря полиморфизму для реализации всех трех типов стеков в C# достаточно создать лишь один общий набор подпрограмм.

6. Основы ООП

В более общем смысле понятие полиморфизма нередко выражается следующим образом: «один интерфейс – множество методов». Это означает, что для группы взаимосвязанных действий можно разработать общий интерфейс.

Полиморфизм помогает упростить программу, позволяя использовать один и тот же интерфейс для описания общего класса действий.

Выбрать конкретное действие (т.е. метод) в каждом отдельном случае – это задача компилятора. Программисту не нужно делать это самому. Ему достаточно запомнить и правильно использовать общий интерфейс.

6. Основы ООП

Представьте, у вас есть классы Кошка и Собака, оба наследуются от базового класса Животное.

У каждого из этих классов есть метод ИздатьЗвук(), который может быть различным для каждого класса:

```
class Животное
{
    Ссылок: 0
    public void ИздатьЗвук()
    {
        Console.WriteLine("Звук животного");
    }
}

Ссылок: 0
class Кошка : Животное
{
    Ссылок: 0
    public void ИздатьЗвук()
    {
        Console.WriteLine("Мяу");
    }
}

Ссылок: 0
class Собака : Животное
{
    Ссылок: 0
    public void ИздатьЗвук()
    {
        Console.WriteLine("Гав");
    }
}
```

6. Основы ООП

С использованием полиморфизма вы можете создать объекты разных классов и вызвать метод ИздатьЗвук() для каждого из них, используя одно и то же имя метода:

```
Животное животное = new Животное();
Кошка кошка = new Кошка();
Собака собака = new Собака();

животное.ИздатьЗвук(); // Выведет: "Звук животного"
кошка.ИздатьЗвук();    // Выведет: "Мяу"
собака.ИздатьЗвук();   // Выведет: "Гав"
```

Здесь мы видим, что даже если метод ИздатьЗвук() имеет одно и то же имя, он ведет себя по-разному для каждого класса. Это и есть полиморфизм - способность объектов разных классов выполнять одну и ту же операцию (в данном случае, метод), но в зависимости от конкретного типа объекта он выполняет разные действия.

Контрольные вопросы

1. Что такое .NET?
2. Что представляет собой Common Type System?
3. Что такое инкапсуляция?
4. Что такое наследование?
5. Что такое полиморфизм?

Список литературы

1. Вагнер Б. C# Эффективное программирование М.: ЛОРИ, 2013. - 320 с.

2. Ишкова Э. А. Самоучитель C#. Начало программирования М.: Наука и техника, 2013. - 496 с.

3. Подбельский В. В. Язык C#. Базовый курс М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

4. Троелсен Э. Язык программирования C# 2010 и платформа .NET4, М.: Москва: Огни, 2016. - 238 с.

ЛЕКЦИЯ ОКОНЧЕНА, СПАСИБО ЗА ВНИМАНИЕ!