1. [methodImpl], StackFrame:

* [MethodImpl] - это атрибут в C#, который используется для управления тем, как метод реализуется и вызывается в CLR (Common Language Runtime). Например, с помощью атрибута [MethodImpl(MethodImplOptions.Synchronized)] можно сделать метод потокобезопасным.
* StackFrame - это класс, который представляет фрейм стека вызовов для метода. Он позволяет получить информацию о методах, которые были вызваны до текущего метода в стеке вызовов.

1. Domain-driven development (DDD), области видимости членов, разница между свойством и полем класса:

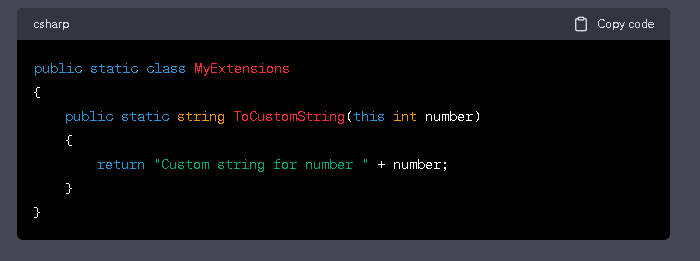
Domain-Driven Design — **подход к проектированию ПО, в основе которого положено тесное сотрудничество клиента и разработчиков**. Заказчик посвящает команду в бизнес-логику своей компании, объясняет, как устроена ее работа, участвует в проектировании сервиса. Основной принцип DDD — разделение приложения на домены.

Домен — предметная область, которая описывает совокупность проблем и целей бизнеса.

* Области видимости членов - это области, в которых определен член класса (поле, свойство, метод). В C# есть 4 уровня областей видимости: public, private, protected и internal.
* Поле класса - это переменная, которая хранит данные объекта. Свойство класса - это специальный метод, который позволяет получать или изменять значение поля. Разница между полем и свойством в том, что свойство позволяет установить дополнительную логику при чтении или записи значения. Get set

1. Разница между абстрактным классом и интерфейсом, наследование и композиция, сигнатура метода, области видимости членов:

* Абстрактный класс - это класс, который содержит хотя бы один абстрактный метод, который должен быть реализован в наследниках. Интерфейс - это набор абстрактных методов и свойств, которые должны быть реализованы классом, который имплементирует интерфейс.
* Наследование - это механизм ООП, который позволяет создавать новый класс на основе существующего, наследуя его поля, свойства и методы.
* Композиция - это механизм ООП, который позволяет создавать объекты, состоящие из других объектов.
* Сигнатура метода - это набор параметров, возвращаемого типа и имени метода. Два метода с одинаковой сигнатурой нельзя объявлять в одном классе.
* Области видимости членов - см. пункт 2.
* 4. Just-In-Time (JIT) компиляция в C# - это процесс компиляции байт-кода CIL (Common Intermediate Language), который генерируется при компиляции исходного кода на C#, в машинный код, который может выполняться на целевой платформе. JIT-компиляция происходит во время выполнения программы, и компилятор JIT-кода может оптимизировать код на основе информации о целевой платформе и способах его использования.
* 5. Common Language Runtime (CLR) - это основная компонента платформы .NET Framework, которая отвечает за управление выполнением программы на управляемых языках, таких как C#. CLR предоставляет механизмы для загрузки и выполнения кода, проверку безопасности, управление памятью и сборку мусора. Управляемые языки отличаются от нативных языков тем, что они компилируются в байт-код, который выполняется на виртуальной машине, а не непосредственно на процессоре. Это позволяет более эффективно управлять памятью и повышает безопасность выполнения программы.
* 6. Сборщик мусора - это компонента CLR, которая автоматически удаляет объекты, которые больше не используются в программе. Сборка мусора происходит автоматически, и программист не должен вручную удалять объекты. Это позволяет избежать ошибок памяти, таких как утечки памяти, и снизить нагрузку на программиста.
* 7. Объект - это экземпляр класса, который имеет свойства и методы, определенные в классе. Объекты используются для хранения данных и выполнения операций над ними. Класс - это шаблон, который определяет свойства и методы объектов, созданных на его основе.
* 8. Статические члены - это свойства и методы, принадлежащие самому классу, а не его экземплярам. Они могут быть вызваны без создания экземпляра класса и используются для работы с данными или функциональностью, которая не зависит от конкретного объекта.
* Экземплярные члены - это свойства и методы, принадлежащие конкретному экземпляру класса (объекту). Они используются для работы с данными или функциональностью, которая зависит от состояния конкретного объекта. Экземплярные члены должны быть вызваны с использованием экземпляра класса.
* Абстрактные члены - это методы или свойства, объявленные в абстрактном классе без реализации. Абстрактные члены должны быть реализованы в классах-наследниках абстрактного класса. Они предоставляют общий контракт, которому должны следовать все классы, наследующие абстрактный класс.
* 9.Дженерики в C# позволяют создавать универсальные типы, которые могут работать с различными типами данных без необходимости явного указания типа. Это позволяет улучшить переиспользуемость кода и уменьшить дублирование кода. Реализуются дженерики в C# с помощью параметризованных типов, которые определяются с использованием символа "<" и ">". Например, List<T> является дженерик-классом, где T - это параметр типа.
* 10.Использование сторонних подключаемых сборок может представлять определенные риски, такие как наличие уязвимостей в коде, неправильное использование API, несовместимость с другими компонентами системы, а также невозможность получения поддержки от автора сборки. Поэтому перед использованием сторонних сборок необходимо проводить тщательную проверку их качества и надежности.
* 11.Преимущества использования сторонних подключаемых сборок заключаются в том, что они могут значительно ускорить разработку программного обеспечения, сократить объем написанного кода, предоставить готовые решения для различных задач. Для подтверждения авторства сборки можно использовать цифровые подписи, сертификаты, информацию об авторе в метаданных сборки.
* 12.Метод расширения - это статический метод, который позволяет добавлять новые методы для уже существующих типов данных без необходимости изменять их исходный код. Методы расширения используются для улучшения читабельности кода, а также для упрощения написания кода. Описать свой метод расширения можно следующим образом:

Этот код добавляет метод ToCustomString для типа int, который возвращает строку с кастомным форматированием числа. Методы расширения можно навешивать на классы, структуры и интерфейсы.

* 13. Да, можно навесить метод расширения на интерфейс. В языке программирования, поддерживающем концепцию расширяемости интерфейсов, такой метод обычно называется методом расширения интерфейса или методом-расширением.
* Метод-расширение - это метод, который не имеет реализации в интерфейсе, но может быть реализован в классах, которые реализуют этот интерфейс
* 14.Интерфейсы в C# не могут содержать приватные члены, потому что все члены интерфейсов по определению являются открытыми для реализации в классах, которые реализуют эти интерфейсы.
* 15.Абстрактные классы в C# могут содержать приватные члены. Абстрактные классы являются классами, которые не могут быть созданы напрямую, а только через наследование. Они используются для создания базовых классов, которые определяют общую функциональность для классов-наследников. Приватные члены в абстрактных классах могут использоваться для внутренней реализации класса и недоступны для наследников.
* 16.Наследование и композиция - это два способа организации кода в объектно-ориентированном программировании. Наследование - это отношение между классами, при котором класс-наследник наследует свойства и методы базового класса. Композиция - это отношение между объектами, при котором один объект содержит ссылку на другой объект как свой член. Разница между наследованием и композицией заключается в том, что при наследовании класс-наследник получает все свойства и методы базового класса, в то время как при композиции объект содержит только ссылку на другой объект и может использовать его методы и свойства только через эту ссылку.
* 17.API (Application Programming Interface) - значит программный интерфейс приложения. В контексте API слово «приложение» относится к любому ПО с определенной функцией. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями
* 18.Flags-enum в C# - это перечисления (enum), которые могут использоваться для определения набора флагов, каждый из которых может быть установлен или сброшен отдельно. Они используются для определения набора флагов, которые могут быть использованы для определения состояния объекта или для передачи нескольких параметров в метод. Размерность flags-enum-ной переменной можно посчитать с помощью метода **Enum.GetValues(typeof(YourEnum)).Length**.

1. Основные члены System.IO: System.IO - пространство имен, которое предоставляет классы, позволяющие работать с файлами, каталогами и другими объектами ввода-вывода. Некоторые основные члены System.IO:

* File: статический класс, который предоставляет методы для работы с файлами, такие как создание, удаление, чтение, запись и т. д.
* Directory: статический класс, который предоставляет методы для работы с каталогами, такие как создание, удаление, переименование и т. д.
* Path: статический класс, который предоставляет методы для работы с путями к файлам и каталогам, такие как объединение путей, получение имени файла и т. д.
* FileStream: класс, который представляет поток для работы с файлом, позволяя читать и записывать данные в файл.

1. Основные члены System.IO.File: System.IO.File - статический класс, который предоставляет методы для работы с файлами. Некоторые основные члены System.IO.File:

* Create: создает файл и возвращает объект FileStream для работы с ним.
* Delete: удаляет указанный файл.
* Exists: определяет, существует ли указанный файл.
* ReadAllText: читает весь текст из указанного файла и возвращает его в виде строки.
* WriteAllText: записывает указанный текст в указанный файл.

1. Назначение StringBuilder, сценарий оптимального применения: StringBuilder - класс, предназначенный для построения строк. В отличие от класса String, который неизменяемый, StringBuilder предоставляет изменяемый объект для построения строки. Оптимально использовать StringBuilder при необходимости построения строки из большого числа элементов, особенно если эти элементы должны быть добавлены в цикле. Это позволяет избежать создания множества объектов String, что может привести к негативному влиянию на производительность приложения.

22.Назовите не менее пяти способов конкатенации строк, преимущества и недостатки:

1.Оператор "+" - позволяет просто объединить две строки. Преимущество - простота использования. Недостаток - при конкатенации большого числа строк это может привести к созданию большого количества временных объектов, что негативно сказывается на производительности.

2.Метод String.Concat - позволяет объединить несколько строк. Преимущество - метод не создает временные объекты, поэтому он более производительный, чем оператор "+". Недостаток - использование метода с большим числом строк

может привести к необходимости передачи массива строк в метод, что может быть неудобно и непрактично.

3. Метод StringBuilder.Append - позволяет добавлять строки к уже существующей строке. Преимущество - класс StringBuilder оптимизирован для построения строк и не создает множество временных объектов, что улучшает производительность. Недостаток - при необходимости конкатенации небольшого числа строк использование StringBuilder может быть избыточным.

4.Метод String.Join - позволяет объединить массив строк в одну строку с разделителем. Преимущество - метод позволяет легко объединять массивы строк, используя определенный разделитель. Недостаток - метод может быть неудобным для конкатенации строк, не находящихся в массиве.

5.Интерполяция строк - позволяет вставлять значения переменных в строку, используя специальный синтаксис. Преимущество - интерполяция строк позволяет создавать более читаемый код, поскольку вставка значений переменных происходит непосредственно в строку. Недостаток - использование интерполяции строк с большим числом переменных может снизить производительность.

23.Способы применения атрибутов в различных фреймворках (не менее трех): Атрибуты - это метаданные, которые позволяют добавлять информацию о типах, членах класса и других элементах программы. Некоторые способы применения атрибутов в различных фреймворках:

1.ASP.NET - атрибуты могут использоваться для настройки поведения веб-страниц и контролов, например, для определения, какую страницу отображать при обращении по определенному URL-адресу.

2.Entity Framework - атрибуты могут использоваться для указания свойств сущности, таких как ключевые поля, связи между таблицами и т. д.

3.NUnit - атрибуты могут использоваться для определения тестовых методов и их атрибутов, таких как тестовый случай, источник данных, ожидаемый результат и т. д.

24.ORM (Object-Relational Mapping) - это технология, которая позволяет связать объектную модель приложения с реляционной моделью данных в базе данных. Она облегчает разработку приложений, позволяя разработчикам работать с объектами и классами в языке программирования, в то время как ORM транслирует эти операции в запросы к базе данных. ORM может автоматически генерировать SQL-запросы и выполнить маппинг объектов на соответствующие записи в базе данных, обеспечивая удобство и быстроту в работе с данными.

25.System.Reflection - это пространство имен в C#, которое содержит классы и методы, предназначенные для работы с метаданными и динамическими операциями во время выполнения программы. Некоторые основные классы в System.Reflection включают Type, MethodInfo, PropertyInfo, FieldInfo, ConstructorInfo и другие. Они позволяют получать информацию о типах данных, методах, свойствах и полях в программе, а также создавать и вызывать эти элементы во время выполнения. Основные методы в System.Reflection включают GetType(), GetMethods(), GetProperties(), GetFields() и другие, которые обеспечивают доступ к метаданным и позволяют выполнять динамические операции с типами данных. System.Reflection также может использоваться для динамической компиляции кода во время выполнения программы.  
26. В C# система типов состоит из двух категорий: ссылочные и значимые типы.

Значимые типы (также известные как структуры) представляют собой типы, значения которых хранятся непосредственно в стеке вызовов, а не в управляемой куче. Значимые типы обычно содержат данные, которые необходимо хранить вместе, например, координаты точки или размеры прямоугольника. Некоторые примеры значимых типов в C# включают в себя int, double, bool и char.

Ссылочные типы (также называемые классами) представляют собой типы, значения которых хранятся в управляемой куче, а не в стеке вызовов. Ссылочные типы обычно содержат данные, которые могут быть распределены по нескольким местам в памяти, например, список или массив объектов. Некоторые примеры ссылочных типов в C# включают в себя строки (string), массивы, списки, классы и интерфейсы.

Различия между ссылочными и значимыми типами в C# включают в себя способ передачи по значению или по ссылке, время жизни и область видимости. Значимые типы передаются по значению, что означает, что копия значения передается методу, в то время как ссылочные типы передаются по ссылке, что означает, что метод получает ссылку на оригинальный объект. Время жизни значимых типов привязано к жизни переменной, в то время как время жизни ссылочных типов привязано к управляемой куче и управляется сборщиком мусора. Область видимости значимых типов ограничена блоком кода, в то время как область видимости ссылочных типов ограничена областью, в которой был создан объект.

{" 27. полиморфизм. compile-time и runtime. Примеры! Являются ли generic-и примером полиморфизма-приведите ход рассуждений"},

Полиморфизм заключается в том, что один фрагмент кода может работать с разными типами данных. В C# это реализуется с помощью перегрузок (overloading).

Compile time - **выбор в коде, во время компиляции** (comple time). То есть, компилятор видит код и понимает, какой тип нужен при создании объекта.

Runtime - Compile time - **выбор в коде, во время компиляции** (comple time). То есть, компилятор видит код и понимает, какой тип нужен при создании объекта.

В общем runtime - это набор чего-то (компонент, библиотек, инструментов), который необходим для работы определенного типа приложения.

Часто под runtime подразумевается механизм выполнения (среда выполнения) .NET приложения, например: CLR, .NET Native или Mono runtime. Этот механизм овечает за управление памятью, загрузку сборок, выполнение, генерацию и компиляцию JIT кода.

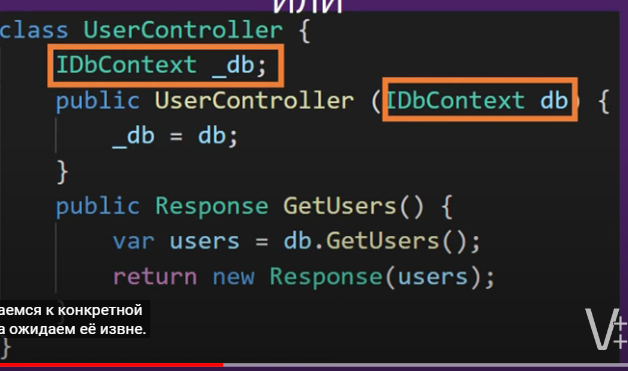
{" 28. dependency injection. Зачем, как. Понятие контейнера зависимости, пример инверсии через конструктор"},

https://www.youtube.com/watch?v=iG1xVFdQwUU

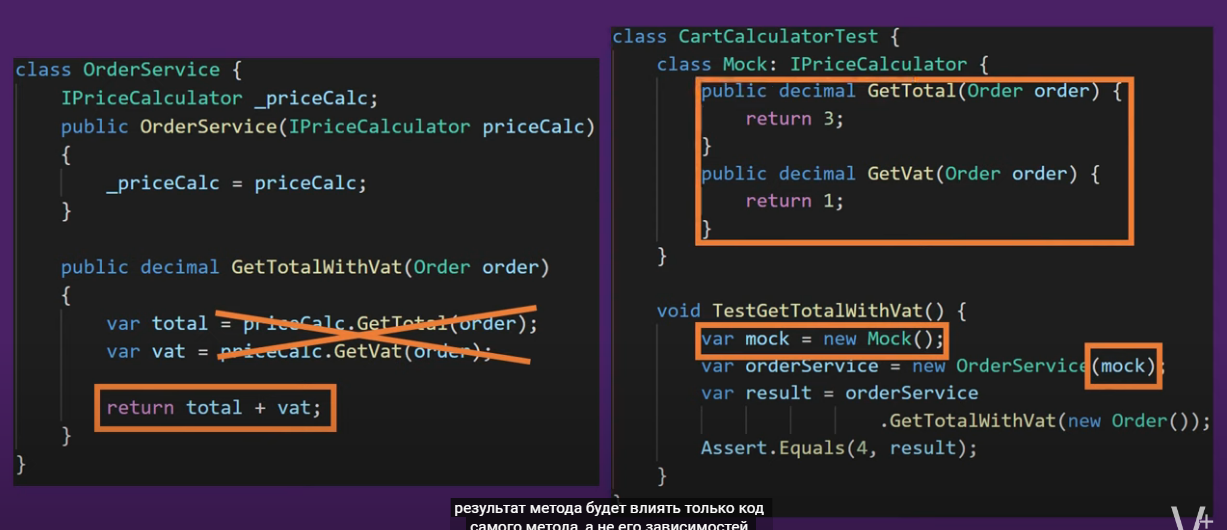
Другой класс, который нужен в текущем

(не надо создавать экземпляр другого класса внутри метода, а передавать его нужно в аргументе)

Набор методов для этого экземпляра определяются интерфейсом

DI контейнер – это контейнер, который хранит список интерфейсов и их реализаций



class Coffe\_Machine

{

IEngin engin;

public Coffe\_Machine(IEngin \_engin)

{

Engin = \_engin;

}

}

Пример из жизни: дисплей ноута – производитель решил за пользователя каким олжен быть дисплей

Решение – подключение внешнего дисплея

Если вы более подробно рассмотрите Dependency Injection (DI), то увидите, что это паттерн проектирования программного обеспечения, который позволяет разрабатывать слабосвязанный код. Через DI вы можете уменьшить «жесткость» связи между программными компонентами. Внедрение зависимостей также известно как Inversion-of-Control (инверсия управления), которая упрощает модульное тестирование.

Крайне важно сделать шаг назад к основам проектирования объектно-ориентированного приложения, где основным аспектом проектирования является «слабая связь». Это означает, что объекты имеют только столько зависимостей, сколько необходимо для выполнения своих заданий, а число зависимостей должно быть ограничено.

Кроме того, зависимости объекта должны быть от интерфейсов, а не от конкретных объектов. Что такое конкретный объект? Это любой объект, созданный с помощью ключевого слова «new».

Полезно понимать Принцип Инверсии Зависимостей ([Dependency Inversion Principle](https://www.oodesign.com/dependency-inversion-principle.html)), который помогает нам при написании слабо связанных классов. Определение:

* Модули высокого уровня не должны зависеть от модулей низкого уровня. Оба должны зависеть от абстракций.
* Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

{" 29. События. Назначение, синтаксис в сишарпе"},

delegate void AccountHandler(string message);

event AccountHandler Notify;

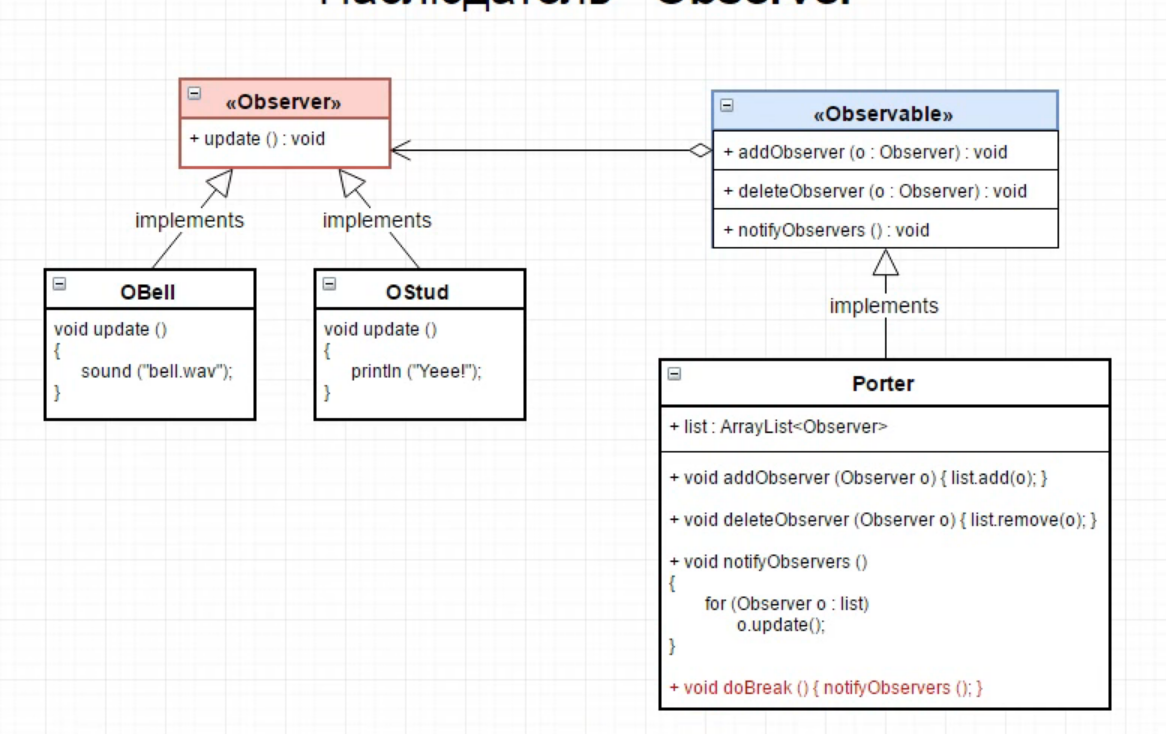
**Событием** в языке C# называется сущность, предоставляющая две возможности: сообщать об изменениях, а для его пользователей — реагировать на них.

{" 30. События. Назначение, пример архитектурной формы"},

Кнопки

{" 31. События. Шаблон проектирования НАБЛЮДАТЕЛЬ - назначение, пример"},

Паттерн "Наблюдатель" (Observer) представляет поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.



Когда использовать паттерн Наблюдатель?

Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях

Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друга от друга.

Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и процессе работы программы может изменяться.

Пример: Школьный звонок – звонок срабатывает и ученики получают оповещение о начале урока

{" 32. Делегаты. Определение, назначение, пример"},

delegate void Message();

static void Main()

{

Message mes; // 2. Создаем переменную делегата

mes = Hello; // 3. Присваиваем этой переменной адрес метода

mes(); // 4. Вызываем метод

**Делегат** в **C#** — это тип, который представляет ссылки на методы со списком параметров и типом возвращаемого значения. **Делегаты** используются для передачи методов в качестве аргументов к другим методам.

{" 33. Инкапсуляция, наследование, полиморфизм. Определение, назначение, примеры

Инкапсуляция– это **скрытие реализации объекта от конечного пользователя**, которое в Си-шарп осуществляется при помощи модификаторов доступа (private, public …). Конечным пользователем объекта здесь выступает либо объект наследник, либо программист. Когда мы объявляем какой-либо член класса приватным, и создаем объект такого класса, доступ к этому члену закрыт для программиста, это и является скрытием реализации.

Class A

{

Private bool Ischange

Private void change()

{

Ischange = true;

}

Public void action()

{

If(!Ischange)

{

Change();

}

Else

{

//действия

Ischange = false;

}

}

Наследование позволяет определить базовый класс для определенных функций (доступа к данным или действий), а затем создавать производные классы, которые наследуют или переопределяют функции базового класса.

Class A : B{}

C# позволяет использовать один метод для разных типов данных и даже переопределить логику операторов.

Console.WriteLine("Hello, World!"); //string

Console.WriteLine(50f); //float

Console.WriteLine(0.23); //double

Console.WriteLine(42); //int