Оглавление

[1. Что такое наследование? Как наследование реализуется в C#? 1](#_Toc69222776)

[2. В чём отличие наследования от агрегации? 1](#_Toc69222777)

[3. Что такое указатель на базовый класс? 2](#_Toc69222778)

[4. Что такое виртуальные функции? Что такое чисто виртуальные функции? 2](#_Toc69222779)

[5. Что такое полиморфизм? Как полиморфизм реализуется в C#? 3](#_Toc69222780)

[6. Какие проблемы могут возникнуть при работе с полиморфными объектами через указатель на базовый класс? 3](#_Toc69222781)

[7. Что такое абстрактный класс? 3](#_Toc69222782)

[8. Что такое статическое поле, статический метод, статический класс? Для чего они нужны? 4](#_Toc69222783)

[9. Что такое свойства? Для чего они нужны? 5](#_Toc69222784)

# 1. Что такое наследование? Как наследование реализуется в C#?

Класс спользует методы, поля, свойства и константы базового класса как свои собственные.

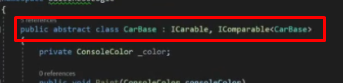
Необходимо чтобы подменять поведение, т. е. использовать полиморфизм.

Не нужно использовать чтобы убирать дублирование, здесь лучше подойдёт агрегация или композиция.

К имени класса прибавляется постфикс Base.

C# и .NET поддерживают только **одиночное наследование**. Это означает, что каждый класс может наследовать члены только одного класса. Но зато поддерживается ***транзитивное наследование***, которое позволяет определить иерархию наследования для набора типов. Другими словами, тип D может наследовать возможности типа C, который в свою очередь наследует от типа B, который наследует от базового класса A. Благодаря транзитивности наследования члены типа A будут доступны для типа D.

Множественное наследование поддерживается при наследовании от интерфейсов, т. е. один класс может быть наследником многих интерфейсов:



Не все члены базового класса наследуются производными классами. Следующие члены не наследуются.

* [Статические конструкторы](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/static-constructors), которые инициализируют статические данные класса.
* [Конструкторы экземпляров](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/constructors), которые вызываются для создания нового экземпляра класса. Каждый класс должен определять собственные конструкторы.
* [Методы завершения](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/destructors), которые вызываются сборщиком мусора среды выполнения для уничтожения экземпляров класса.

Все остальные члены базового класса наследуются производными классами, но их видимость не зависит от доступности. Доступность членов влияет на видимость для производных классов следующим образом.

* [Закрытые](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/private) члены являются видимыми только в производных классах, которые вложены в базовый класс.
*  [Защищенные](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/protected) члены являются видимыми только в производных классах.
*  [Внутренние](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/internal) члены являются видимыми только в производных классах, которые находятся в той же сборке, что и базовый класс.
* [Открытые](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/public) члены являются видимыми в производных классах, а также входят в общедоступный интерфейс производных классов.

# 2. В чём отличие наследования от агрегации?

Агрегация – включение одного класса в другой класс.

Пример агрегации - передача в конструктор одного класса экземпляра другого класса и присвоение полю первого класса экземпляр(уже созданного извне) другого класса.

При наследовании класс- наследник имеет все свойства базового класса, при агрегации – класс содержит поля или свойства своих составных частей

Если новый класс является более или менее оригинальным классом. Используйте наследство. Новый класс теперь является подклассом исходного класса.

Если новый класс должен иметь исходный класс. Используйте агрегацию. Новый класс теперь имеет оригинальный класс в качестве члена.

Короче говоря. Мы должны использовать агрегацию, если часть интерфейса не используется или должна быть изменена, чтобы избежать нелогичной ситуации. Нам нужно только использовать наследование, если нам нужны почти все функциональные возможности без серьезных изменений. И если есть сомнения, используйте агрегацию.

# 3. Что такое указатель на базовый класс?

При использовании наследования указатель или ссылка на тип базового класса способен адресовать объект любого производного от него класса. Возможность манипулировать такими указателями или ссылками независимо от фактического типа адресуемого объекта называется полиморфизмом.

Во время выполнения объекты производного класса могут обрабатываться как объекты базового класса в таких местах, как параметры метода и коллекции или массивы.

Базовые классы могут определять и реализовывать [виртуальные](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/virtual) методы, а производные классы — [переопределять](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/override) их, т. е. предоставлять свое собственное определение и реализацию. Во время выполнения, когда клиент вызывает метод, CLR выполняет поиск типа объекта во время выполнения и вызывает перезапись виртуального метода. В исходном коде можно вызвать метод в базовом классе и обеспечить выполнение версии метода, относящейся к производному классу.

# 4. Что такое виртуальные функции? Что такое чисто виртуальные функции?

***Виртуальным*** называется такой метод, который объявляется как **virtual** в базовом классе. Виртуальный метод отличается тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода. Кроме того, виртуальные методы интересны тем, что именно происходит при их вызове по ссылке на базовый класс. В этом случае средствами языка C# определяется именно тот вариант виртуального метода, который следует вызывать, исходя из типа объекта, к которому происходит обращение по ссылке, причем это делается во время выполнения. Поэтому при ссылке на разные типы объектов выполняются разные варианты виртуального метода. Иными словами, вариант выполняемого виртуального метода выбирается по типу объекта, а не по типу ссылки на этот объект.

Так, если базовый класс содержит виртуальный метод и от него получены производные классы, то при обращении к разным типам объектов по ссылке на базовый класс выполняются разные варианты этого виртуального метода.

чисто виртуальная функция - это функция без реализации. Делает класс, в котором присутствует, абстрактным (т.е. таким, объекты которого нельзя создать, а можно создать только указатели и ссылки на него). Наследуемые классы, которые не определяют чисто виртуальную функцию базового, также становятся абстрактными.

# 5. Что такое полиморфизм? Как полиморфизм реализуется в C#?

Переопределение поведения в наследниках и использование по указателю(ссылке) на базовый класс.

Полиморфизм- замещение поведения в классе наследника. В ООП используется динамический полиморфизм. (Статический полиморфизм. Пример- Перегрузка функции, перегрузка оператора)

Динамический полиморфизм

C # позволяет создавать абстрактные классы, которые используются для обеспечения частичной реализации класса интерфейса. Реализация завершается, когда производный класс наследуется от него. Абстрактные классы содержат абстрактные методы, которые реализуются производным классом. Производные классы имеют более специализированную функциональность.

# 7. Что такое абстрактный класс?

**Абстрактные** классы содержат абстрактные методы, которые реализуются производным классом. Производные классы имеют более специализированную функциональность.

Вот правила об абстрактных классах:

* Вы не можете создать экземпляр абстрактного класса
* Вы не можете объявить абстрактный метод вне абстрактного класса
* Когда класс объявляется **запечатанным** , его нельзя унаследовать, абстрактные классы не могут быть объявлены герметичными.

# 8. Что такое статическое поле, статический метод, статический класс? Для чего они нужны?

СТАТИЧЕСКОЕ ПОЛЕ ЭТО ЗЛО и нужно бить по рукам за его использование (прямая цитата!)

Если у статического класса есть состояние это боль, потому что оно доступно и меняется все время жизни. При тестировании класса нельзя добиться его изолированности, т. к. не создаётся внутри класса. С классом мы понимаем что было выполнен

Мы не можем гарантировать, сколько вызовов было до тестирования, нельзя зафиксировать состояние при тестировании статических классов.

Из-за того, что статический класс живет все время, мы точки обращения к статическому классу не можем контролировать (на самом деле можем, но использование глобального состояние усложняет жизнь). Применять использование глобального состояния нужно когда мы понимаем риск, когда это обосновано (пример – хранение настроек программы, делаем глобальное состояние и обращаемся к ним из разных участков программы, стараясь не изменять их).

СТАТИЧЕСКИЙ МЕТОД

С точки зрения области видимости данных, которые вы изменяете с помощью этого статического метода.

Public и private статические методы это хорошо

Когда есть статический метод, из-за того что все остальное нестатическое, когда мы обращаемся к статике, у неё четко область видимость, она работает со статическими методами, данными , либо с тем что в неё передали, область видимости не весь класс, а только этот метод. В него что-то пришло, там обработалось и ретёрном вернулось, и нам не нужно думать о валидации и некорректному присвоению данных, здесь четкие входы и выходы. Понятный вход и выход. Не нужно смотреть на весь класс, только на метод, если там не используются статические поля.

[Статический](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/static) класс в основном такой же, как и нестатический класс, но имеется одно отличие: нельзя создавать экземпляры статического класса. Другими словами, нельзя использовать оператор [new](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/new-operator) для создания переменной типа класса. Поскольку нет переменной экземпляра, доступ к членам статического класса осуществляется с использованием самого имени класса. Например, если есть статический класс, называемый UtilityClass, имеющий открытый статический метод с именем MethodA, вызов метода выполняется, как показано в следующем примере:

Статический класс может использоваться как обычный контейнер для наборов методов, работающих на входных параметрах, и не должен возвращать или устанавливать каких-либо внутренних полей экземпляра.

Статические поля, методы, свойства относятся ко всему классу и для обращения к подобным членам класса необязательно создавать экземпляр класса

Если весь класс является статическим:

* Нельзя создавать экземпляр класса, используя ключевое слово new.
* Не разрешается использовать не статические члены этого же класса.
* Он не поддерживает наследование.
* Невозможно перегрузить методы.

Если класс не является статическим, но содержит статические методы, то на эти методы распространяются следующие ограничения:

* Не разрешается использовать не статические члены этого же класса из статических. Конечно же, вам никто не мешает создать экземпляр класса в статическом методе.
* Наследование и полиморфизм для статических членов не поддерживаются.

Итак: **статический конструктор вызывается перед доступом к любому члену класса, за исключением констант**.

# 9. Что такое свойства? Для чего они нужны?

Кроме обычных методов в языке C# предусмотрены специальные методы доступа, которые называют свойства. Они обеспечивают простой доступ к полям классов и структур, узнать их значение или выполнить их установку.

Здесь у нас есть закрытое поле name и есть общедоступное свойство Name. Хотя они имеют практически одинаковое название за исключением регистра, но это не более чем стиль, названия у них могут быть произвольные и не обязательно должны совпадать.

Через это свойство мы можем управлять доступом к переменной name. Стандартное определение свойства содержит блоки get и set. В блоке get мы возвращаем значение поля, а в блоке set устанавливаем. Параметр value представляет передаваемое значение.

Возможно, может возникнуть вопрос, зачем нужны свойства, если мы можем в данной ситуации обходиться обычными полями класса? Но свойства позволяют вложить дополнительную логику, которая может быть необходима, например, при присвоении переменной класса какого-либо значения. Например, нам надо установить проверку по возрасту

Мы можем применять модификаторы доступа не только ко всему свойству, но и к отдельным блокам - либо get, либо set