# **Для чего используют статические классы?**

Статические классы в C# представляют собой классы, которые не могут быть инстанциированы (то есть, у них нельзя создать объект), и все их члены и методы являются статическими.

* экземпляры такого класса создавать запрещено
* не должен реализовывать никаких интерфейсов (не вызвать)
* от него запрещено наследовать
* все элементы такого класса должны явным образом объявляться с модификатором static
* может иметь статический конструктор
* Компилятор не создает автоматически конструктор по умолчанию

В C#, статические классы используются для создания классов, члены которых принадлежат самому классу, а не экземпляру класса. Это означает, что вы можете вызывать методы и использовать поля статического класса, не создавая экземпляр этого класса.

Вот несколько основных причин использования статических классов в C#:

1. Если у вас есть методы или свойства, которые не зависят от конкретного экземпляра объекта, вы можете поместить их в статический класс.
2. Статические классы также могут содержать статические поля, которые общие для всех экземпляров класса.
3. Статические классы часто используются для создания утилитарных классов, предоставляющих набор методов для решения определенных задач.

# **2. Что может содержать статический класс?**

Статический класс в C# может содержать следующие элементы:

1. \*\*Статические поля (переменные):\*\* Это переменные, которые принадлежат классу, а не конкретному экземпляру класса.

2. \*\*Статические методы:\*\* Это методы, которые принадлежат классу и могут быть вызваны без создания экземпляра класса.

3. \*\*Статические свойства:\*\* Это свойства, которые также принадлежат классу и могут быть использованы без создания экземпляра.

4. \*\*Статические события:\*\* Это события, которые привязаны к классу, а не к конкретному экземпляру.

5. \*\*Статические вложенные классы:\*\* Внутри статического класса можно объявлять другие статические классы.

6. \*\*Статические конструкторы:\*\* Статический класс может иметь статический конструктор, который выполняется при первом обращении к классу.

# **3. Что такое производный и базовый классы?**

Базовый класс — это класс, от которого наследуются другие классы. Он является источником для общих свойств и методов, которые могут быть использованы в производных классах. Базовый класс обычно создается для описания общих черт или функциональности, которая должна быть доступна для нескольких типов объектов.

Производный класс — это класс, который наследует свойства и методы от базового класса. Он может также добавлять свои уникальные свойства и методы или переопределять поведение унаследованных членов. Производный класс использует общую функциональность базового класса и расширяет ее или модифицирует под конкретные требования.

# **4. Как используют ключевое слово base?**

Ключевое слово `base` в языке программирования C# используется для доступа к членам базового класса из производного класса. Вот несколько основных сценариев, в которых используется `base`:

1. \*\*Вызов методов базового класса:\*\*

2. \*\*Использование базового конструктора:\*\*

3. \*\*Использование в переопределенных свойствах:\*\*

# **5. В чем заключена основная задача наследования?**

Наследование − это механизм получения нового класса на основе уже существующего

* формирует иерархию
* поощряет повторное использование кода

Правила наследования:

* 1) В C# наследование всегда подразумевается открытым
* 2) Запрещено множественное наследование классов (но не интерфейсов)
* 3) наследуются все свойства, методы, поля и т.д., которые есть в базовом классе
* 4) Производному классу доступны public, internal, protected и protected internal
* 5) не наследуются конструкторы базового класса
* 6) тип доступа к производному классу должен быть таким же, как и у базового класса или более строгим

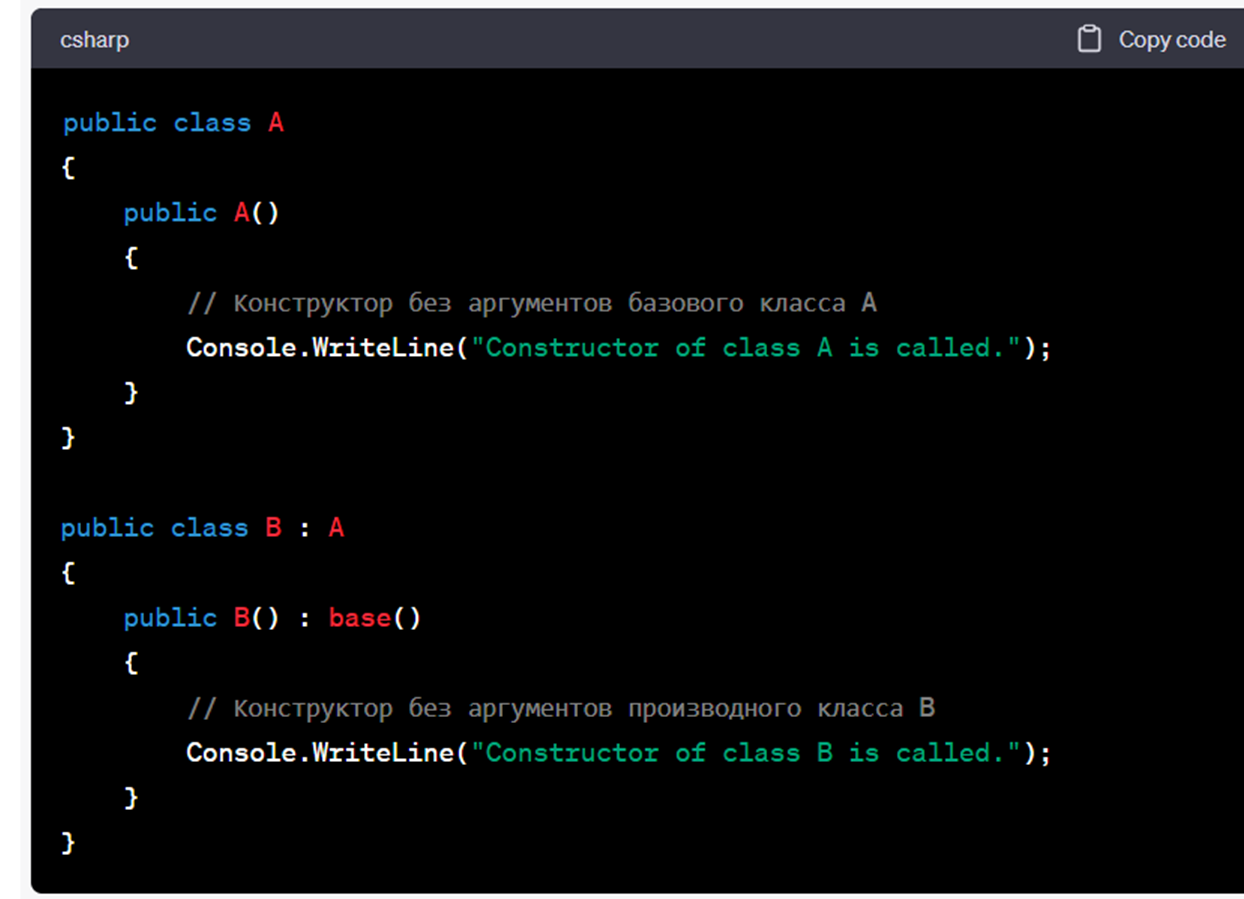
# **6. Пусть базовый класс содержит метод basefunc(), а производный класс не имеет метода с таким именем. Может ли объект производного класса иметь доступ к методу basefunc()? Если да, то при каких условиях?**

Если базовый класс содержит метод `basefunc()`, а производный класс не имеет метода с таким же именем, объект производного класса не может напрямую вызывать метод `basefunc()` базового класса, так как этот метод не является членом производного класса.

Однако доступ к методу `basefunc()` может быть осуществлен через ключевое слово `base` внутри производного класса при условии, что метод `basefunc()` в базовом классе имеет модификатор доступа, который позволяет доступ из производного класса (например, `protected` или `public`).

# **7. Напишите объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов базового класса A.**

В C# конструктор производного класса может вызывать конструктор базового класса с использованием ключевого слова `base`. Вот пример объявления конструктора без аргументов для производного класса `B`, который вызывает конструктор без аргументов базового класса `A`:



# **8. Что такое полиморфизм? Приведите пример.**

Полиморфизм - это свойство объектов в программировании, которое позволяет им вести себя по-разному в зависимости от контекста использования. Проще говоря, это способ использовать один и тот же метод или свойство для объектов разных типов.

Например, у разных животных может быть одинаковый метод "издать звук", но каждый вид животных будет издавать свой уникальный звук.

* -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

Статический и динамический полиморфизм - это две различные формы полиморфизма в объектно-ориентированных языках программирования. Вот основные различия между ними:

- \*\*Статический полиморфизм:\*\* Также известный как раннее (или статическое) связывание. Решение о том, какая версия метода будет вызвана, принимается на этапе компиляции. Примерами являются перегрузка методов (overloading) и шаблоны (в C++), где компилятор на основе сигнатур и контекста решает, какой метод использовать.

- \*\*Динамический полиморфизм:\*\* Также известный как позднее (или динамическое) связывание. Решение о том, какой метод вызвать, принимается во время выполнения, что позволяет более гибкую замену объектов и использование интерфейсов. Примерами являются виртуальные методы и интерфейсы, где вызываемый метод определяется во время выполнения в зависимости от типа объекта.

Необходимые условия для реализации позднего связывания:

* классы должны образовывать иерархию наследования;
* в классах должны быть методы с одинаковой сигнатурой. Элементы (методы) производных классов должны перекрывать (override) соответствующие элементы (методы) базовых классов;
* элементы (методы) класса должны быть виртуальными, то есть должны быть обозначены ключевыми словами virtual, override.

|  |  |  |
| --- | --- | --- |
| **Тип связывания** | **Достоинства** | **Недостатки** |
| Ранее | высокое быстродействие получаемых выполнимых программ | снижение гибкости программ |
| Позднее | высокая гибкость выполняемой программы, возможность реакции на события | относительно низкое быстродействие программы |

# **9. Определите назначение виртуальных функций**.

- Главная цель виртуальных функций — обеспечить динамический полиморфизм.( Решение о том, какой метод вызвать, принимается во время выполнения)

- Виртуальные функции позволяют объектам различных классов использовать один и тот же интерфейс (название и сигнатура метода), что делает возможным вызывать методы, принадлежащие производным классам, через указатель (или ссылку) на базовый класс.

- Виртуальные функции могут быть переопределены в производных классах. Это позволяет классам-наследникам предоставлять свою собственную реализацию методов, унаследованных от базового класса.

Правила переопределения

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

# **10. Кому доступны переменные с модификатором protected?**

В языке программирования C#, переменные с модификатором `protected` доступны следующим образом:

1. \*\*В самом классе:\*\*

2. \*\*В производных классах:\*\*

3. \*\*Внутри того же класса в другом экземпляре:\*\*

# **11. Наследуются ли переменные с модификатором private?**

Переменные с модификатором `private` не наследуются в классах-наследниках в языке программирования C#. Модификатор `private` обозначает, что член класса виден только в пределах того класса, где он был объявлен, и не доступен в производных классах.

# **12. As, is – что это, как применяется? В чем между ними отличие ?**

В языке программирования C#, ключевые слова `as` и `is` используются для работы с типами данных, особенно в контексте проверки и приведения типов.

- `as` используется для попытки приведения типа в случае, если объект может быть преобразован в указанный тип.

- Если преобразование успешно, возвращается объект указанного типа; в противном случае, возвращается `null`.

- `as` часто используется совместно с проверкой на `null` для безопасного приведения типов.

\*\*`is`:\*\*

- `is` используется для проверки, является ли объект экземпляром указанного типа.

- Возвращает `true`, если объект является экземпляром указанного типа, и `false` в противном случае.

- `is` часто используется в условных выражениях для проверки типа объекта перед его приведением.

# **13. Поддерживает ли C# множественное наследование?**

\*\*Множественное наследование\*\* - это возможность класса наследовать атрибуты и методы сразу от нескольких базовых классов. Это означает, что класс может быть производным от нескольких родительских классов.

Нет, C# не поддерживает множественное наследование классов. В C# класс может наследовать только один базовый класс.

Однако C# поддерживает множественное наследование интерфейсов, что позволяет классу реализовать несколько интерфейсов. Это обеспечивает гибкость в определении поведения объектов без введения проблем, связанных с множественным наследованием классов.

# **14. Можно ли запретить наследование от класса?**

В C# можно использовать ключевое слово `sealed`, чтобы запретить наследование от класса. Когда класс объявлен с использованием `sealed`, он не может быть использован в качестве базового класса для других классов.

# **15. Можно ли разрешить наследование класса, но запретить перекрытие метода?**

В C# ключевое слово `sealed` используется для запрета переопределения методов в производных классах. Однако, если вы хотите разрешить наследование класса, но предотвратить переопределение конкретного метода, то можно сделать метод `sealed`, даже если сам класс не является `sealed`.

# **16. Что такое абстрактный класс?**

Абстрактный класс в языке программирования C# — это класс, который не может быть создан напрямую (экземпляр абстрактного класса не может быть создан), и предназначен для службы в качестве базового класса для других классов. Абстрактные классы часто используются для определения базовой функциональности, которую должны реализовать производные классы.

* Служит только для порождения потомков - предоставляют базовый функционал для классов-наследников.
* Задает интерфейс для всей иерархии
* Может содержать и полностью определенные методы, переменные, конструкторы, свойства
* 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:

* Методы
* Свойства
* Индексаторы
* События

# **17. В каком случае вы обязаны объявить класс абстрактным?**

1. \*\*Если класс содержит хотя бы один абстрактный метод:\*\* Абстрактные методы являются методами без реализации в самом классе. Если ваш класс содержит хотя бы один абстрактный метод, то вам следует объявить класс как абстрактный.

2. \*\*Если класс не предоставляет полную реализацию всех унаследованных абстрактных методов:\*\* Если ваш класс наследуется от другого абстрактного класса, который содержит абстрактные методы, ваш класс должен предоставить реализацию всех этих абстрактных методов.

3. \*\*Если класс является базовым для других классов, и предполагается, что он будет расширен другими классами:\*\* Абстрактные классы часто используются в качестве базовых классов для других классов. Если вы хотите, чтобы ваш класс служил только в качестве базового класса и не мог быть использован напрямую для создания экземпляров, объявление класса как абстрактного является хорошей практикой.

# **18. В чем разница между абстрактными и виртуальными классами? Между виртуальными и абстрактными методами?**

- \*\*Абстрактный метод:\*\* Не имеет реализации в базовом классе. Объявляется с использованием ключевого слова `abstract`.

- \*\*Виртуальный метод:\*\* Имеет реализацию по умолчанию в базовом классе, но может быть переопределен в производных классах. Объявляется с использованием ключевого слова `virtual`.

# **19. Какие компоненты класса могут быть виртуальными?**

В языке программирования C#, ключевое слово `virtual` может быть применено к методам и свойствам, делая их виртуальными. Виртуальные члены класса могут быть переопределены в производных классах, что предоставляет возможность изменять или расширять их поведение.

1. \*\*Виртуальные методы:\*\* Виртуальные методы предоставляют реализацию по умолчанию в базовом классе, которую можно переопределить в производных классах.

2. \*\*Виртуальные свойства:\*\* Виртуальные свойства позволяют переопределять геттеры и сеттеры в производных классах.

3. \*\*Виртуальные индексы (indexers):\*\* Виртуальные индексы позволяют переопределять индексаторы в производных классах.

# **20. Что такое интерфейс?**

интерфейс представляет собой абстрактный контракт, определяющий методы, которые класс или структура обязаны реализовать. Интерфейсы предоставляют способ определения общего интерфейса для нескольких классов без необходимости использования множественного наследования.

Основные черты интерфейсов в C#:

1. Интерфейс сам по себе не предоставляет реализацию методов, он только объявляет сигнатуры методов, свойств, индексов и событий.

2. Классы, реализующие интерфейс, обязаны предоставить реализацию всех методов, определенных в интерфейсе.

3. Класс может реализовывать несколько интерфейсов, предоставляя таким образом функциональность, определенную в каждом интерфейсе.

|  |  |
| --- | --- |
| **Свойства** | **Интерфейс** |
| не может содержать | Не статические поля, операции, конструкторы, деструкторы, типы, |
| может содержать | абстрактные методы, обобщения свойства и индексаторы, а также события, делегаты  Default методы,  Статические поля, методы (с реализацией) и константы (с С# 8.0) |
| Доступность методов | **publiс по умолчанию (не указывается )**  (при переопределении тоже public) Могут быть private и protected |
| наследуются | С# поддерживается одиночное наследование для классов и множественное — для интерфейсов  (при реализации интерфейса нужно обеспечить точное совпадение)  Сначала всегда указывается имя базового класса, затем указывается интерфейс |
| Расширение интерфейса | Интерфейс наследуется интерфейсом |
| Имена | с прописной буквы I (не обязательно) |

# **21. Что может содержать интерфейс?**

1. Методы
2. Свойства:
3. Индексы (indexers)
4. События:\*\*
5. Другие интерфейсы:

# **22. Как работать с объектом через унаследованный интерфейс?**

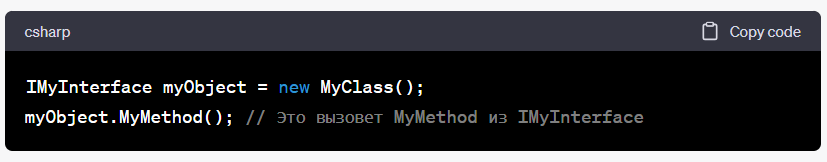
Работа с объектом через унаследованный интерфейс в C# позволяет использовать общий набор методов и свойств, определенных в интерфейсе, независимо от конкретного типа объекта.

# **23. Приведите пример явной реализации интерфейса.**

1. Неявная реализация интерфейса: Когда класс реализует интерфейс неявно, вы просто предоставляете реализацию методов интерфейса как обычных методов в теле класса.

### 2. Явная реализация интерфейса: Когда вы явно реализуете интерфейс, методы интерфейса реализуются с указанием имени интерфейса. Это позволяет избежать конфликтов имен, когда класс реализует несколько интерфейсов с одинаковыми методами.

При явной реализации интерфейса вы должны вызывать методы через переменную типа интерфейса:



\*\*Когда использовать явную реализацию интерфейса:\*\*

1. \*\*Когда класс реализует несколько интерфейсов с одинаковыми методами:\*\*

- Явная реализация позволяет избежать конфликтов имен.

2. \*\*Когда вы хотите скрыть реализацию от внешнего кода:\*\*

- Явная реализация делает методы интерфейса недоступными из экземпляра класса.

# **24. Почему нельзя указать модификатор видимости для методов интерфейса?**

В C# методы интерфейса не могут иметь модификаторы видимости, потому что интерфейс предоставляет только контракт, определяя сигнатуры методов, но не их реализацию. Методы интерфейса автоматически считаются публичными и не могут иметь модификаторы доступа (public, private, protected, internal) или другие модификаторы, такие как static или virtual.

# **25. Можно ли наследовать от нескольких интерфейсов?**

Да, в C# класс может наследовать от нескольких интерфейсов. Это называется множественным наследованием интерфейсов. В отличие от множественного наследования классов, множественное наследование интерфейсов поддерживается в C# и является одним из ключевых моментов объектно-ориентированного программирования.

# **26. Назовите отличия между интерфейсом и абстрактным классом.**

**Определение методов:**

Абстрактный класс может содержать как абстрактные методы (методы без реализации), так и конкретные методы (методы с реализацией). Подклассы обязаны реализовать все абстрактные методы или могут наследовать реализацию конкретных методов.

Интерфейс может содержать только абстрактные методы, то есть методы без какой-либо реализации. Классы, которые реализуют интерфейс, обязаны предоставить реализацию всех методов интерфейса.

|  |  |  |
| --- | --- | --- |
|  | **Абстрактные классы** | **Интерфейсы** |
|  | одна иерархия | несколько иерархий |
| модификаторы доступа | задаются | publiс не явно, могут быть другие модификаторы |
| Поля | есть | Могут быть |
| Наследник | может не определять – абстрактный | Должен реализовывать все элементы |
| В списке предков может быть | один абстр. класс | несколько интерфейсов |
| Объекты создавать | нельзя | нельзя |
| набор действий | имеет смысл только для конкретной иерархии | к различным иерархиям |

# **27. Для чего используются стандартные интерфейсы ICloneable, IComparable, IComparer, lEnumerable?**

### 1. `ICloneable` (Клонирование): Используется для создания копии объекта.- \*\*Метод:\*\* `Clone()`

### 2. `IComparable` (Сравнение): Используется для реализации сравнения объектов.- \*\*Метод:\*\* `CompareTo()`

### 3. `IComparer` (Сравнение для нескольких объектов): Используется для создания внешнего компаратора для объектов, когда реализация `IComparable` невозможна или нежелательна.- \*\*Метод:\*\* `Compare()`

### 4. `IEnumerable` и `IEnumerator` (Итерация по коллекциям): Используется для предоставления возможности перебора элементов в коллекции. - \*\*Методы:\*\* `IEnumerable` имеет метод `GetEnumerator()`, который возвращает `IEnumerator`, а `IEnumerator` имеет методы `MoveNext()`, `Reset()` и свойство `Current`.