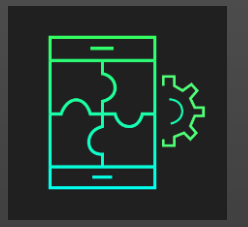
БАЗОВЫЙ КЛАСС

class Трансформер



Поля (Fields):
- Хранят данные, связанные с классом.
- Определяются в теле класса и могут иметь различные модификаторы доступа (например, private, public).

Свойства (Properties):
- Обеспечивают доступ к полям через методы чтения и записи, позволяя добавлять логику при получении и установке значений.

Методы (Methods):
- Определяют поведение класса, могут принимать параметры и возвращать значения.
- Методы могут быть статическими (относятся к классу, а не к экземпляру) или экземплярными.

Конструкторы (Constructors):
- Специальные методы, автоматически вызываемые при создании объекта класса.
- Используются для инициализации полей класса.

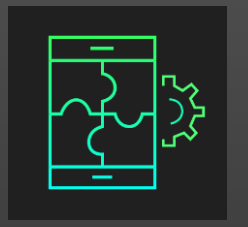
Деструкторы (Destructors):
- Вызываются при уничтожении объекта класса для освобождения ресурсов, не поддерживается в C# напрямую, вместо этого обычно используются IDisposable и паттерн "Dispose".

Наследование:
- Классы могут наследоваться от других классов, что позволяет переиспользовать код и создавать иерархии классов.

Интерфейсы (Interfaces):
- Классы могут реализовывать интерфейсы, что позволяет определять контракты для классов.

КЛАСС-НАСЛЕДНИК

class Бамблби : Трансформер



Наследование в C# — это механизм, позволяющий одному классу (наследнику) получать свойства и методы другого класса (родителя). Это обеспечивает повторное использование кода и создание иерархий классов.

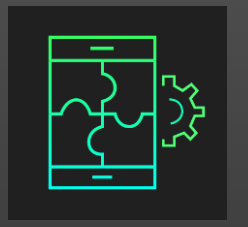
Родительский класс (Base Class):
- Класс, от которого наследуются свойства и методы.
- Может быть объявлен с использованием модификатора доступа public, protected или internal.

Наследник (Derived Class):
- Класс, который наследует функциональность родительского класса.
- Может добавлять свои уникальные члены и переопределять родительские методы.

Переопределение методов (Method Overriding):
- Наследник может изменять реализацию метода родительского класса, используя ключевое слово override при объявлении метода в родительском классе с модификатором virtual или abstract.

Модификаторы доступа:
- При наследовании важно правильно указать модификаторы доступа для определения видимости членов классов (например, protected делает члены доступными только для наследников и в пределах того же сборки).

ЭКЗЕМПЛЯР КЛАССА



```
var трансформ = new Трансформер()
```

Экземпляр класса — это конкретный объект, созданный на основе определенного класса. Он имеет свои уникальные значения полей и может использовать методы и свойства, определенные в классе

Инициализация:
Экземпляры классов создаются с помощью оператора new, который выделяет память под объект и вызывает конструктор класса.

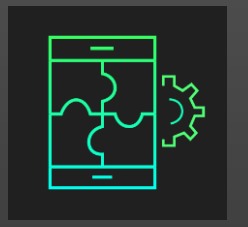
Состояние и поведение:
Каждый экземпляр имеет собственное состояние, представляемое значениями его полей. Поведение экземпляров определяется методами, определенными в классе.

```
Бамблби бамблби = new() // экземпляр класса-наследника
{
    Имя = "Бамблби",
    МодельАвто = "Chevrolet Camaro",
    Спецспособность = "Включить радио"
};

var трансформ = new Трансформер() // экземпляр основного класса
{
    Имя = "ОптимусПрайм",
    МодельАвто = "Грузовик",

};
```

ПОЛИМОРФИЗМ



Полиморфизм — это возможность объектов разных классов использовать один и тот же метод, что позволяет одному методу обрабатывать разные типы объектов. В C# он достигается через наследование и перегрузку методов. Один из принципов ООП (Объектно-Ориентированного Программирования)

Преимущества полиморфизма:

•Обеспечивает повышенную гибкость кода.
•Позволяет уменьшить дублирование кода.
•Упрощает поддержку и расширение в дальнейшем.
•Позволяет создавать более абстрактные и обобщенные интерфейсы.

Состояние и поведение:

Каждый экземпляр имеет собственное состояние, представляемое значениями его полей. Поведение экземпляров определяется методами, определенными в классе.

Использование интерфейсов:

Полиморфизм часто достигается через использование интерфейсов. Разные классы могут реализовывать один и тот же интерфейс, обеспечивая единый интерфейс для взаимодействия с различными реализациями.

Типы полиморфизма:

В C# существуют два основных типа полиморфизма:

- Компиляционный полиморфизм (статический): достигается с помощью перегрузки методов и операторов. Метод может иметь одинаковое имя, но разное количество или тип параметров.
- Исполнительный полиморфизм (динамический): достигается через переопределение виртуальных и абстрактных методов. Это позволяет вызывать методы, которые определяются в подклассах, в зависимости от типа объекта во время выполнения.

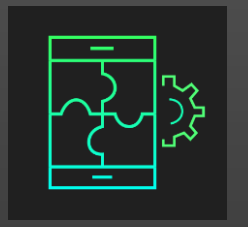
Объявление виртуальных и абстрактных методов:

Для достижения динамического полиморфизма методы должны быть объявлены как virtual в базовом классе и переопределены в производном классе с использованием ключевого слова override.

Безопасность типов:

Полиморфизм в C# сохраняет безопасность типов, что означает, что при использовании полиморфных методов типы объектов проверяются на этапе компиляции.

НАСЛЕДОВАНИЕ



Наследование в C# — это принцип ООП (Объектно-Ориентированного Программирвоания), позволяющий создавать новые классы на основе существующих. Производный класс наследует свойства и методы базового класса и может добавлять свои собственные. Это способствует переиспользованию кода (коммунальности) и упрощает его поддержку.

Определение:

Наследование позволяет одному классу (производному или подклассу) унаследовать элементы (поля, свойства, методы) другого класса (базового или родительского). Это нужно для создания иерархии классов и обеспечения лучшей структуры кода.

Типы наследования:

- **Одноуровневое (simple)**: Один класс наследует от одного базового класса.
- **Многоуровневое (multilevel)**: Один класс наследует от другого класса, который сам является производным от базового.
- **Множественное наследование**: C# не поддерживает множественное наследование для классов (нельзя унаследовать от нескольких классов), однако это возможно с помощью интерфейсов.

Сигнатуры методов:

При наследовании методы родительского класса могут быть переопределены в производном классе, используя ключевое слово override, что позволяет изменять их поведение.

Доступность элементов:

Элементы базового класса могут иметь различные уровни доступа: public, protected, internal, private. Например, доступ к защищенным (protected) элементам возможен только из производных классов.

КЛЮЧЕВОЕ СЛОВО var

var

var — ключевое слово в C#, используемое для неявной типизации переменной, позволяющее компилятору определить тип на основе присваиваемого значения.

Автоматическое выведение типа
Когда используется var, компилятор определяет тип переменной в момент компиляции, основываясь на типе значения, присваиваемого переменной. Это делает var особенно полезным в случаях, когда тип значения может быть длинным или сложным

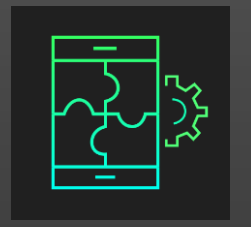
Только для инициализированных объектов:
Переменные, объявляемые с использованием var, должны обязательно быть инициализированы сразу же в момент объявления. Невозможно объявить переменную без начального значения.

Читаемость кода:
var позволяет улучшить читаемость кода, особенно при работе с длинными или вложенными типами (например, типами с обобщениями) и анонимными типами.

```
// было
Трансформер трансформ = new Трансформер()

//стало
var трансформ = new Трансформер()
```

КЛЮЧЕВОЕ СЛОВО virtual



virtual

Ключевое слово virtual в C# объявляет метод или свойство, который может быть переопределен в производных классах, что позволяет реализовать полиморфизм.

Определение

Метод или свойство, обозначенное как virtual, предоставляет базовую реализацию, которую можно изменить в наследующих классах.

Необходимо для:

Для создания общей функциональности в базовом классе и кастомизации ее в производных классах.

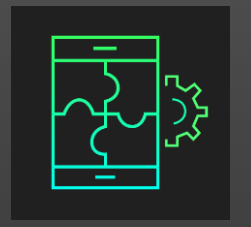
С чем используется:

Ключевое слово virtual часто используется вместе с override, чтобы переопределять методы из базовых классов в производных.

Полиморфизм:

Когда используется virtual, компилятор позволяет применять динамический выбор метода во время выполнения, в зависимости от типа объекта, к которому применяется вызываемый метод.

КЛЮЧЕВОЕ СЛОВО override



override

override — это ключевое слово в C#, которое позволяет переопределить виртуальный или абстрактный метод базового класса

Определение

Ключевое слово override применяется в производном классе, чтобы изменить поведение метода, который был объявлен как virtual или abstract в базовом классе. Это позволяет производному классу использовать тот же метод, но с другим функционалом.

Сигнатура метода:

При переопределении метода необходимо сохранить ту же сигнатуру (имя, тип возвращаемого значения и параметры), что и у метода в базовом классе.

Виртуальные и абстрактные методы:

Для того чтобы применить override, метод в базовом классе должен быть объявлен с использованием ключевого слова virtual (метод с реализацией) или abstract (метод без реализации).

Динамический полиморфизм:

override позволяет реализовать динамический полиморфизм, когда метод вызывается на основании типа объекта во время выполнения, а не на основании статического типа переменной.