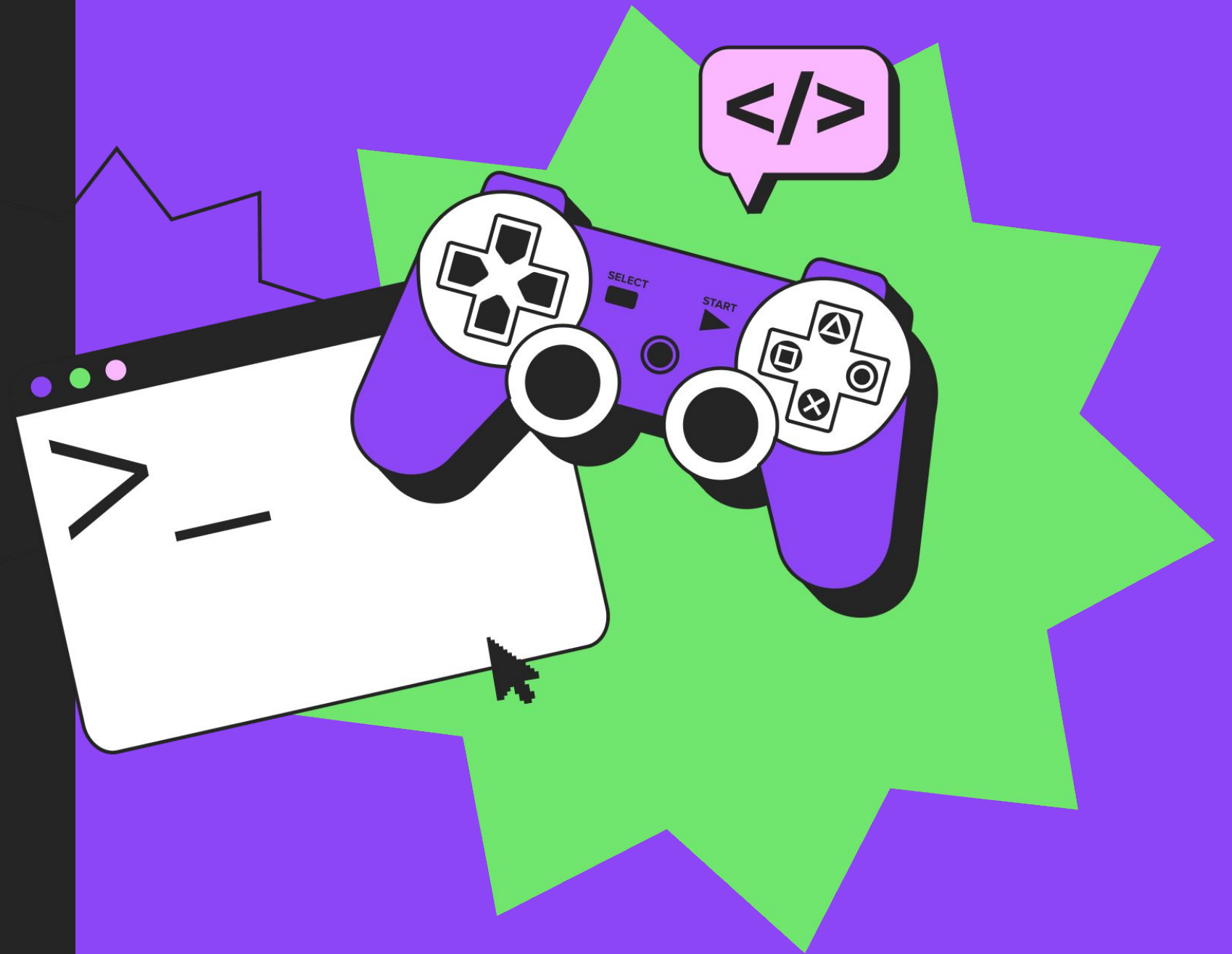


# Разработка приложения на C#

Урок 1  
Классы и ООП





# План курса

1

Классы и ООП

2

Интерфейсы и обобщения

3

Коллекции (1 часть)

4

Коллекции (2 часть)

5

Делегаты и события

6

Исключения

7

Рефлексия

8

Работа с файловой  
системой: потоки и  
буферизация

9

Сериализация

# Содержание урока



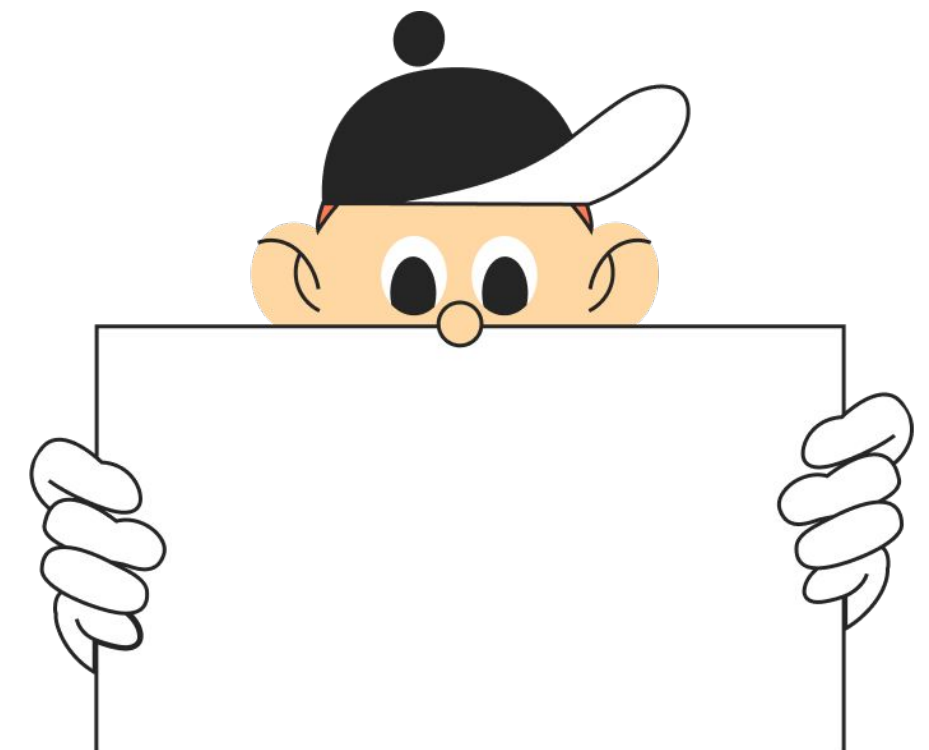
На этой лекции вы узнаете про:

- ООП (SOLID, KISS, DRY)
- Классы C#
- Наследование
- Модификаторы доступа
- Абстрактные классы
- Статические методы
- Статические классы
- Методы расширения



# ООП

**ООП** или **объектно-ориентированное программирование** — это методология программирования, в основе которой лежит объект — совокупность данных (полей и свойств), а также действий (методов), объединённых в одну сущность.





# Принципы ООП

ООП строится на 3 основных принципах:

- **Инкапсуляция** — способность объекта скрывать свою внутреннюю реализацию, объединяя в себя данные и методы.
- **Наследование** — возможность создавать новые объекты (наследник/child) на основе уже существующих (родитель/parent). Объекты-потомки наследуют поведение и свойства объектов-родителей.
- **Полиморфизм** — способность выполнять действия с объектом-потомком так, как если бы он был базовым объектом.



## ООП: SOLID

- **S** — single responsibility
- **O** — open-closed principle
- **L** — Liskov substitution principle
- **I** — interface segregation
- **D** — dependency inversion

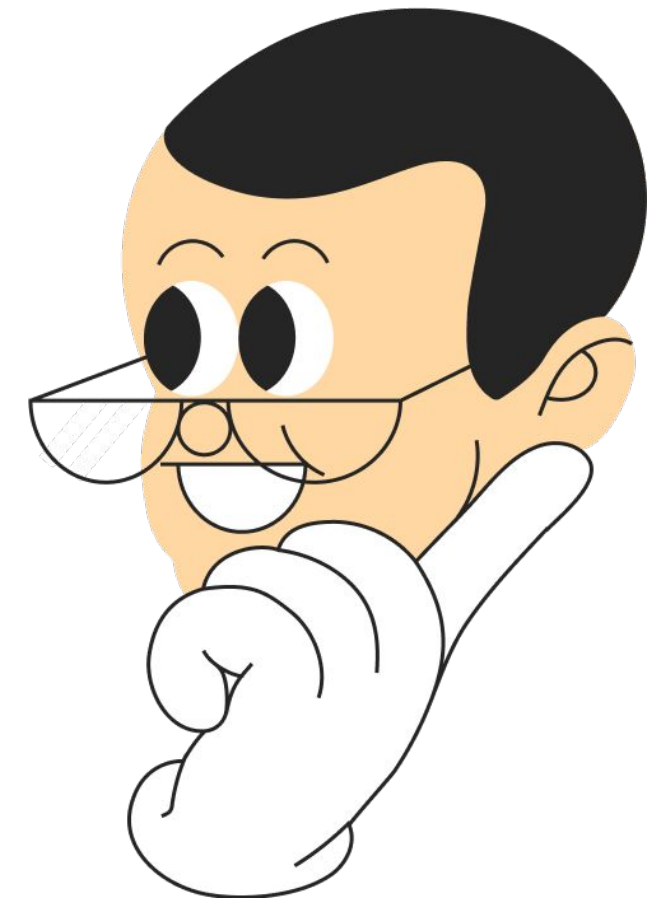


# ООП: KISS

**Keep It Short and Simple**



Код должен быть прост и понятен. Не нужно использовать средства более сложные, чем это необходимо для решения задачи, как бы привлекательно это ни выглядело.





# ООП: DRY

**Don't Repeat Yourself**



Не повторяйтесь — если у вас есть два одинаковых куска кода в программе, подумайте, как сделать так, чтобы остался только один.







# Class

**Класс** — тип данных, модель для создания объектов определенного типа, описывающая данные (поля) и алгоритмы для его работы (методы). Экземпляр класса называется объектом.



В C# класс стоит во главе всей системы типов: все классы C# происходят от базового типа `object`.



В C# любое приложение является классом.



# Class

```
[модификатор доступа] class Идентификатор[:Родитель] [,Интерфейс]

{

    //поля, свойства методы

}
```



# Class

**Обязательными при объявлении класса являются:**

- ключевое слово `class`;
- идентификатор (имя) класса, следующий за `class`;
- фигурные скобки.



Для создания экземпляра класса используется оператор `new`, идущий перед именем класса.



# Поля класса

```
class ИмяКласса  
{  
    [модификатор доступа] тип имя [= инициализация];  
}
```



# Конструкторы класса

```
class ИмяКласса
{
    [модификатор доступа] ИмяКласса()
    {
        код конструктора1
    }

    [модификатор доступа] ИмяКласса(тип имя)
    {
        код конструктора2
    }
}
```



# Методы класса

```
[модификатор доступа] ИмяКласса  
  
{  
  
    [модификатор доступа] тип возвращаемого значения Имя1 ()  
  
    {  
  
        код метода1  
  
    }  
  
    [модификатор доступа] тип возвращаемого значения Имя2 (параметры)  
  
    {  
  
        код метода2  
  
    }  
  
}
```



# Свойства класса

**Свойства класса** — это члены класса, реализующие различные способы доступа к полям класса.





## Свойства класса

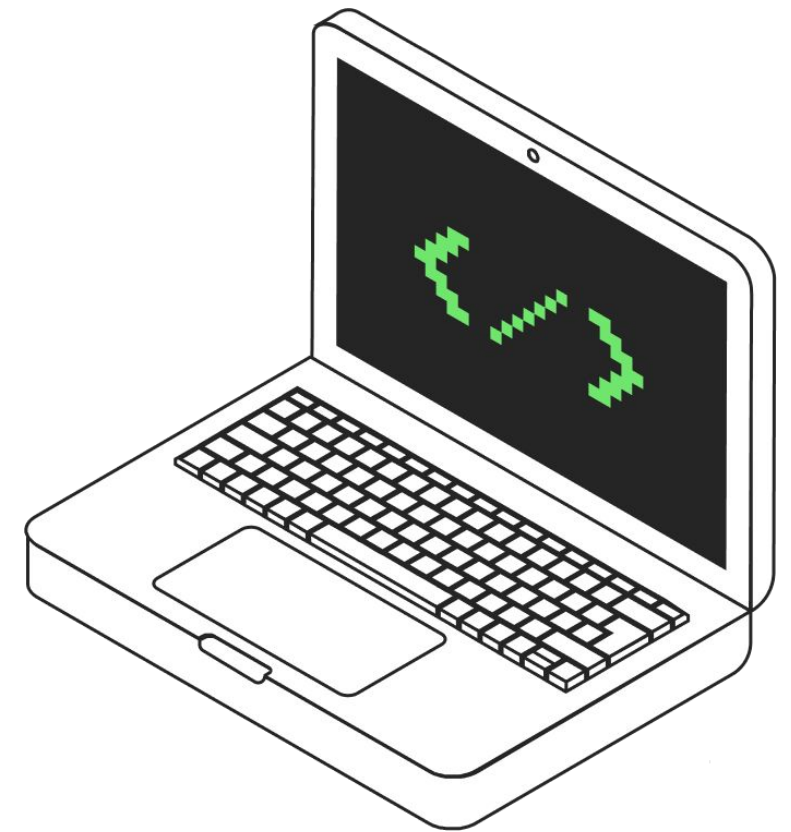
```
[модификатор доступа] Тип ИмяСвойства  
  
{  
  
    [модификатор доступа]    get;  
  
    [модификатор доступа]    set/init;  
  
}
```





# Наследование

```
class ИмяКласса: ИмяКлассаРодителя  
{  
  
}
```





## Безопасное приведение типов объектов, оператор `as`

`тип имя_переменной1 = имя_переменной2 as тип`

или

`var имя_переменной1 = имя_переменной2 as тип`



## Условный оператор ?

```
Person person = null;
```

```
person.Print() //приведет к ошибке
```

```
person?.Print() //не выполнится, не приводя к ошибке
```

```
var father = Person.Father //приведет к ошибке
```

```
var mother = Person?.Father //значение mother будет равным null
```



# Переопределение методов

**Переопределение методов** — это механизм изменения поведения базовых классов в классах-потомках.

Переопределены могут быть методы, свойства, события и индексаторы.



# Переопределение методов

```
class ИмяКлассаРодителя
{
    virtual тип ИмяСвойства;
    virtual тип ИмяМетода()
    {
        код метода
    }
}
class ИмяКлассаПотомка:ИмяКлассаРодителя
{
    override тип ИмяСвойства;
    override тип ИмяМетода()
    {
        код метода
    }
}
```



## Вызов методов базового типа

Из классов-потомков всегда можно получить доступ к методам базового класса, несмотря на то, что они были переопределены в потомке. Для этого существует ключевое слово **base**.





## Модификаторы доступа

- **public** — к типу или члену, помеченному `public`, можно получить доступ из любого другого кода текущей или другой сборки.
- **private** — тип или член, помеченный словом `private`, доступен только коду текущего класса или структуры.
- **protected** — тип или член, помеченный словом `protected`, доступен только коду текущего класса или коду классов-наследников.
- **internal** — к типу или члену, помеченному `internal`, можно получить доступ из любого другого кода текущей, но не другой сборки.
- **protected internal** — к типу или члену, помеченному `protected internal`, можно получить доступ из любого другого кода текущей сборки или из классов потомков, объявленных в других сборках.
- **private internal** — тип или член, помеченный словом `private internal`, доступен только коду текущего класса или коду классов-наследников текущей сборки.



# Абстрактные классы

```
[модификатор доступа] abstract ИмяКласса  
  
{  
  
    [модификатор доступа] abstract тип имяМетода();  
  
    [модификатор доступа] abstract тип имяСвойства{get;set;}  
  
}
```





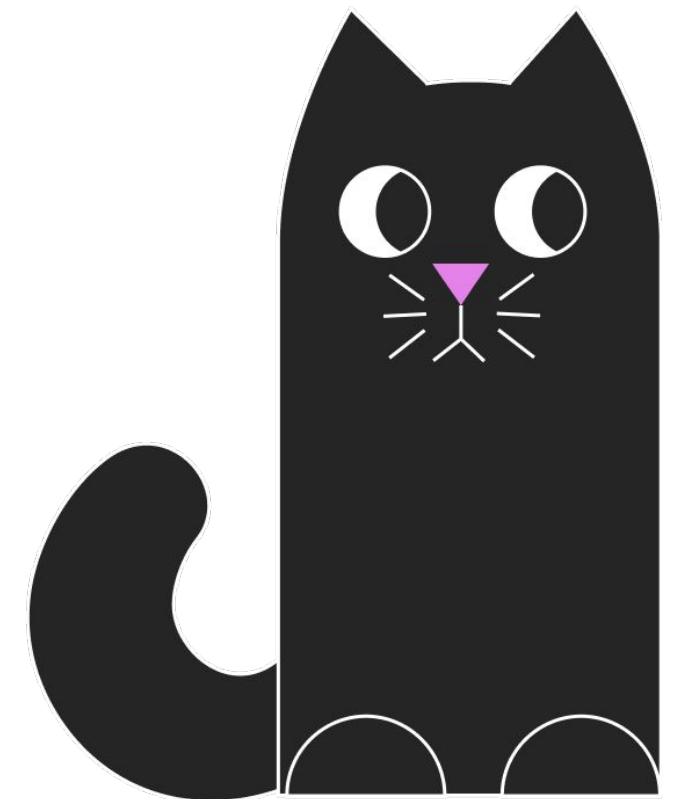
## Запечатанные классы и методы

```
[модификатор доступа] sealed ИмяКласса  
  
{  
  
    [модификатор доступа] sealed тип имяМетода();  
  
    [модификатор доступа] sealed тип имяСвойства { get; set; }  
  
}
```



## Статические методы

Статические методы не принадлежат экземпляру класса и не имеют доступа к членам его экземпляров. Статические методы удобно использовать для реализации определенного функционала, который не связан с конкретным экземпляром класса.





## Статические классы

```
[модификатор доступа] static ИмяКласса  
  
{  
  
    [модификатор доступа] static тип имяПеременной;  
  
    [модификатор доступа] static тип имяМетода ();  
  
    [модификатор доступа] static тип имяСвойства {get;set;}  
  
}
```



# Методы расширения


```
[модификатор доступа] static тип имяМетода (this тип имяПараметра);
```

# Подведение итогов



На этой лекции вы:

- Рассмотрели основы ООП (SOLID, KISS, DRY)
- Исследовали классы C#
- Изучили, что такое:
  - Наследование
  - Модификаторы доступа
  - Абстрактные классы
  - Статические методы
  - Статические классы
  - Методы расширения

**Спасибо**   
**за внимание**

