# ЧЕТВЕРТОЕ ЗАНЯТИЕ

# ООП

## Объектно-ориентированное программирование

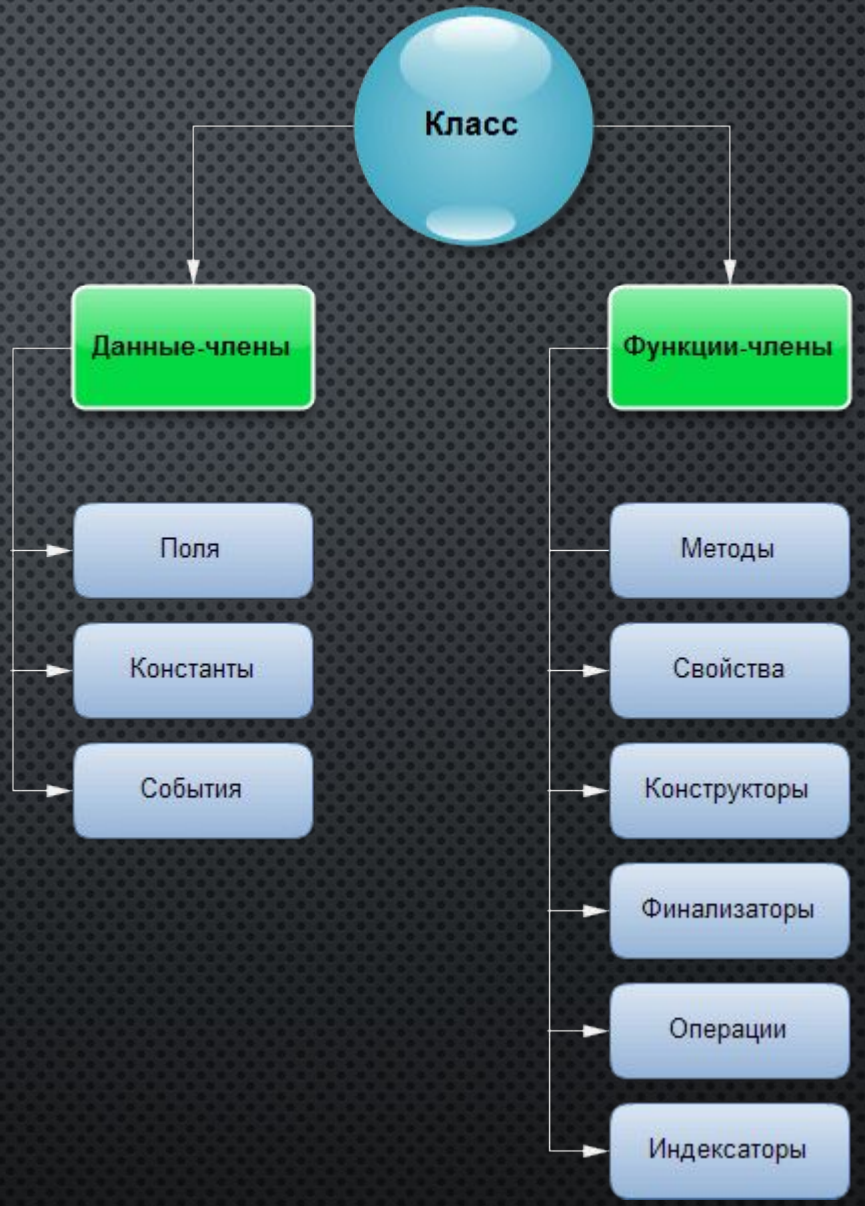
# Принципы ООП

- Инкапсуляция
  - Как данный язык скрывает детали внутренней реализации объектов и предохраняет целостность данных?
- Наследование
  - Как данный язык стимулирует многократное использование кода?
- Полиморфизм
  - Как данный язык позволяет трактовать связанные объекты сходным образом?
- Абстракция
  - Как данный язык позволяет отделять способы использования объектов от их конкретной реализации?

# КЛАССЫ И ОБЪЕКТЫ

# КЛАСС

**Класс** представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов, которые являются экземплярами класса.



# ПРОСТЕЙШИЙ КЛАСС

```
class sample
{

}
```

# Создание экземпляра (объекта)

```
Sample sampleVar = new Sample();
```

Поля класса

```
class Book
{
    public string name;
    public string author;
    public int year;
}
```

# Метод для работы с данными

$ используется для подстановки значений

```
class Book
{
    ...
    public void Info()
    {
        Console.WriteLine(
            $"Book {name} author {author} year - {year}");
    }
}
```

# КОНСТРУКТОР. ПРЕДНАЗНАЧЕН ДЛЯ ИНИЦИАЛИЗАЦИИ ПЕРЕМЕННЫХ

ВАЖНО!
Конструктор не имеет возвращаемого типа!
Даже не void

```
class Book
{
    …
    public Book()
    {
    }
    public Book(string Name, string Author, int year)
    {
        name = Name;
        author = Author;
        year = Year;
    }
}
```

# Статичные поля

Если нам необходимо определить поведение не для конкретного объекта книги, а для всего класса "Книга" мы можем использовать ключевое слово static. Если мы определяем поля, методы или свойства как статические, то они существую не на уровне конкретного объекта, а на уровне всего класса.

Таким образом мы можем обращаться  к ним так:

<Название класса>.<Статическое поле/метод/свойство>

# Модификаторы доступа

- public: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
- private: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.
- protected: такой член класса доступен из любого места в текущем классе или в производных классах.
- internal: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
- protected internal: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.

# Свойства

В то время, как класс хранит в своих полях необходимые ему данные, он не должен предоставлять внешнему коду прямой доступ к ним. Он должен как-о ограждать свои данные либо методами либо **свойствами**

# Определение

Есть приватное поле

```
private int field;
```

Только класс, в котором это поле объявлено, имеет доступ к данному полю

Чтобы предоставить доступ к полю, мы можем обернуть данное поле в свойство

```
public int Field { get{ return field;} set{ field = value;} }
```

Важно понимать, что свойство - это просто пара обычных методов, оформленных более элегантно. Не более. И кроме строки для возвращения значения и присваивания мы можем писать что угодно

# Авто свойства

Для удобства, если нам не надо совершать какие либо действия кроме изменения значения самого поля мы можем использовать автоматические свойства.

```
public int Field {get; set;}
```

Такой код сам сгенерирует приватное поле для хранения значения и методы, которые этим полем манипулируют.

Позже, если понадобится, мы сможем изменить логику поведения данного свойства на необходимую нам. При этом мы не затронем внешний код, который пользуется нашим классом.