

# Лекция 5

Структуры и классы

ah-h!



ARROWS KEYS-MOVE  
ENTER-FIRE



FIRE



OLD-GAMES.RU

Как хранить состояние отдельного корабля?

Несколько несвязанных переменных:

```
int Ship1Size = 1;
```

```
ShipState Ship1State = ShipState.Alive;
```

```
int Ship1X = 2;
```

```
int Ship1Y = 4;
```



# Недостатки такого подхода

- Нельзя хранить в массиве и обрабатывать в цикле
- Много переменных и легко ошибиться
- Спагетти-код

Решение - ООП

# Что такое ООП

- Все есть объект
- Объект - черный ящик для окружающих
- Имеющий публичный интерфейс

# Плюсы

- Структурированность
- Модульность
- Распараллеливаемость разработки

# Минусы

- Сложность для новичков
- Многословность и громоздкость кода
- Излишнее усложнение при плохой архитектуре

# Из чего состоят объекты

- Данные (в том числе другие объекты)
- Внутренняя реализация
- Публичный интерфейс

## Данные:

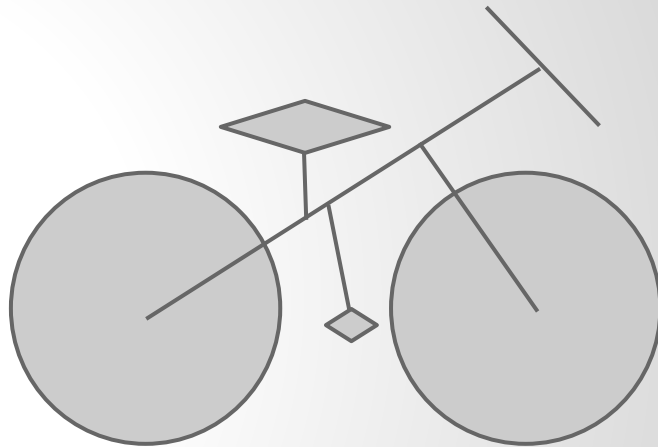
- Текущая передача
- Направление
- Скорость

## Реализация:

- Тормоза
- Система передач

## Интерфейс:


- Руль с переключателями
- Седло
- Педали





# Данные

- Строки с текстом
- Целые и дробные числа
- Объекты
- Структуры



Константы и  
переменные

# Реализация

- **Функции обрабатывающие данные**  
(из каденса и передаточного числа получить скорость)
- **Функции обрабатывающие внешние воздействия**  
(при нажатии на тормозную ручку сжать колодки)

# Интерфейс

Строго описанная спецификация по взаимодействию с объектом:

- Руль отвечает за поворот
- Тормозная ручка отвечает за торможение
- Кручение педалей приводит к движению велосипеда

# Средства ООП в С#

- Структуры - struct
- Классы - class
- Интерфейсы - interface

# Структуры

```
struct Ship {  
    public ShipState state;  
    public int size;  
    public int x;  
    public int y;  
}
```

```
Ship[] ships = new Ship[10];  
ships [0].size = 1;  
ships [0].state = ShipState.Alive;  
ships [0].x = 4;  
ships [0].y = 3;
```

# Структуры

```
struct Student {  
    public int age;  
    public string name;  
    public string group;  
    ...  
}
```

```
Student stud;  
stud.age = 20;  
stud.name = "Василий";  
stud.group = "ИТС-21";
```

# Структуры и классы

```
struct StudentStruct {  
    public int age;  
    public string name;  
    public string group;  
    ...  
}
```

```
class StudentClass {  
    public int age;  
    public string name;  
    public string group;  
    ...  
}
```

# Структуры и классы

```
public static void Main (string[] args)
{
    StudentStruct Ivan;
    Ivan.Name = "Иван";
    Console.WriteLine (Ivan.Name);

    StudentClass Petr;
    Petr.Name = "Петр";
    Console.WriteLine (Petr.Name);
}
```

Структуры  
создаются на  
стеке (stack)

Классы создаются  
в куче (heap)



# Создание объекта

```
class StudentClass {  
    public string Name;  
};  
public static void Main (string[] args)  
{  
    StudentClass Petr = new StudentClass();  
    Petr.Name = "Петр";  
    Console.WriteLine (Petr.Name);  
}
```

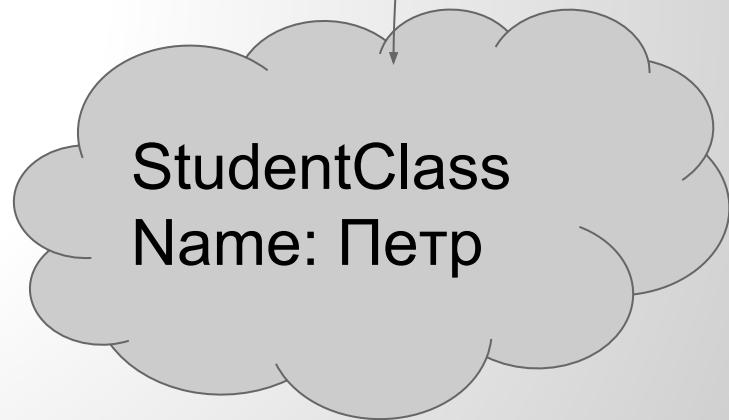
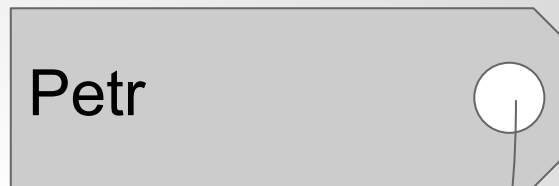
StudentClass

The diagram illustrates the relationship between a C# class and an object instance. A gray rectangular box labeled 'StudentClass' with a white circle on its right side represents the class. A curved arrow originates from the 'new' keyword in the code 'new StudentClass()' and points to the 'StudentClass' box. Another curved arrow originates from the white circle on the 'StudentClass' box and points to a gray cloud shape. Inside the cloud, the text 'Petr' and 'Name: Петр' are displayed, representing an instance of the class.

Petr  
Name: Петр

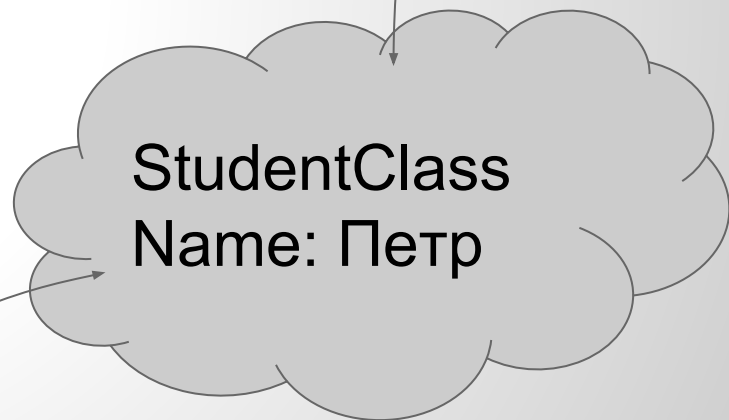
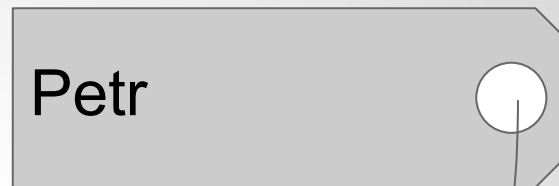
# Ссылки

```
StudentClass Petr = new StudentClass();  
Petr.Name = "Петр";
```



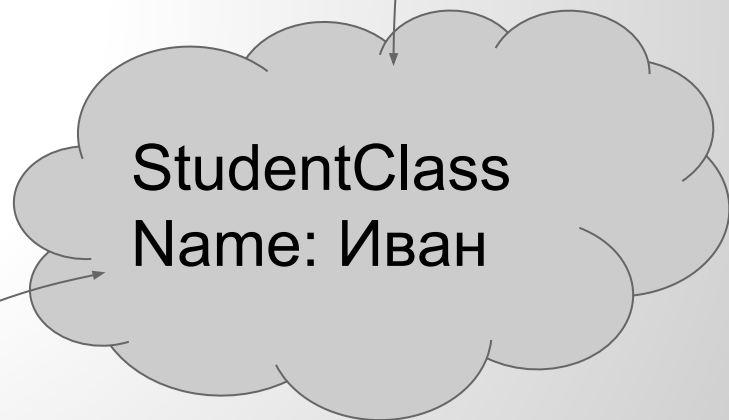
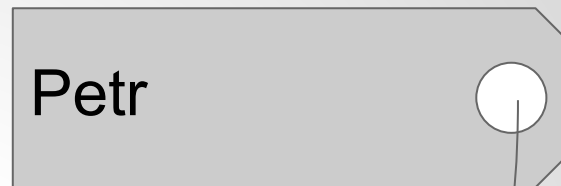
# Ссылки

```
StudentClass Petr = new StudentClass();  
Petr.Name = "Петр";  
StudentClass Ivan = Petr;
```



# Ссылки

```
StudentClass Petr = new StudentClass();  
Petr.Name = "Петр";  
StudentClass Ivan = Petr;  
Ivan.Name = "Иван";  
Console.WriteLine (Petr.Name);
```



# Ссылки

Stud



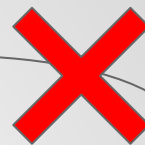
The diagram illustrates the relationship between a variable and an object. A grey rectangular tag with a white circle on its right side contains the text 'Stud'. A curved arrow originates from this circle and points to a grey cloud shape. Inside the cloud, the text 'StudentClass' and 'Name: Петр' is displayed, representing an instance of the StudentClass.

StudentClass  
Name: Петр

```
StudentClass Stud;  
Stud = new StudentClass();  
Stud.Name = "Петр";  
Console.WriteLine (Stud.Name);
```

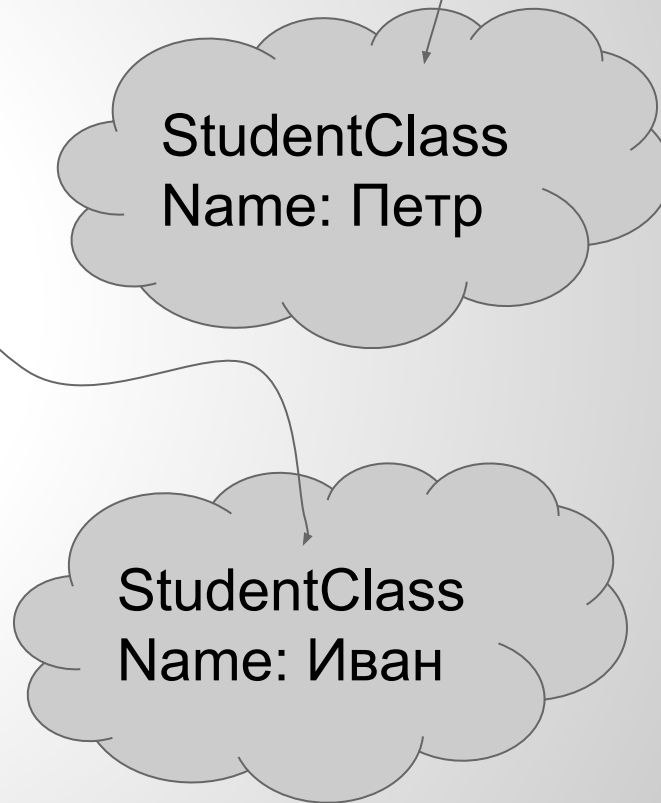
# Ссылки

Stud



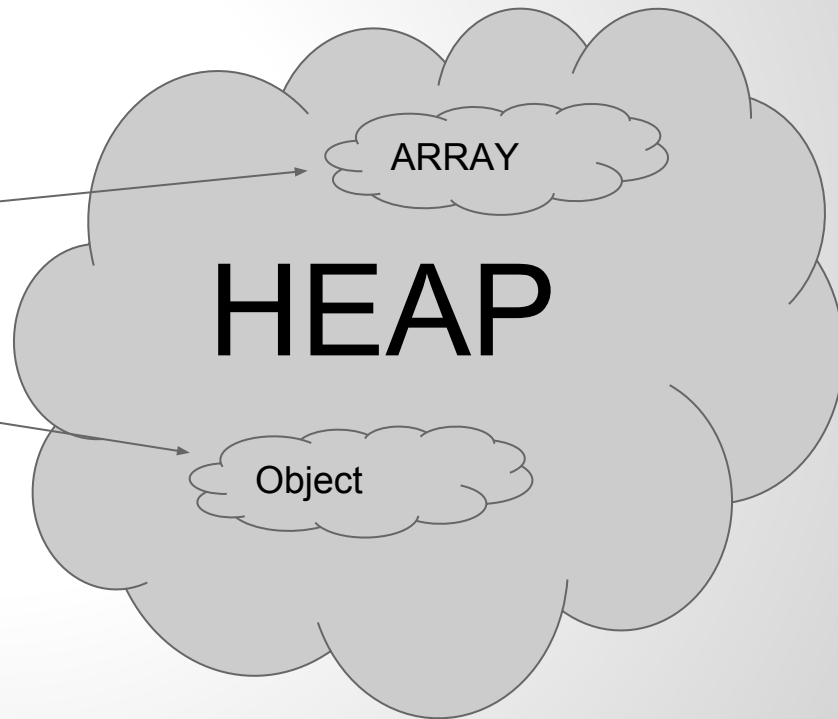
```
StudentClass Stud;  
Stud = new StudentClass();  
Stud.Name = "Петр";  
Console.WriteLine (Stud.Name);
```

```
Stud = new StudentClass();  
Stud.Name = "Иван";  
Console.WriteLine (Stud.Name);
```



# Значимые и ссылочные типы value & reference types

INT	10
DOUBLE	3.1415...
INT[]	array[10]
STRUCT	Ship
CLASS	Object
STACK	



# struct vs class

- Используем структуру если
  - Небольшой размер
  - Простое поведение
  - Нет необходимости в наследовании
- Используем классы если
  - Размер больше 16 байт
  - Сложная инициализация и поведение
  - Требуется наследование



# Объектно-ориентированный подход

- Разбиваем программу на логически связанные сущности - Юнит, Корабль, Снаряд и т.д.
- Указываем в классе данные которые описывают этот объект - сила, здоровье, скорость
- Описываем методы, которые класс выполняет - переместиться, нанести урон

# Данные (поля данных) объекта

```
class Student {  
    public string Name;  
    private int Age;  
};  
public static void Main (string[] args)  
{  
    Student stud = new Student();  
    stud.Name = "Василий";  
    stud.Age = 20;  
}
```

# Контроль данных класса

```
public class Person {  
    public int age;  
}  
  
public static void Main (string[] args)  
{  
    Person pers = new Person();  
    pers.age = -20;  
}
```

# Используя сеттеры и геттеры

```
public class Person {  
    private int m_age;  
    public int age(){  
        return m_age;  
    }  
    public void setAge(int age){  
        if (age > 0 && age < 120) {  
            m_age = age;  
        }  
    }  
}
```

# Используя свойства

```
public class Person {  
    private int m_age;  
    public int age {  
        set {  
            if (value > 0 && value < 120) {  
                m_age = value;  
            }  
        }  
        get {  
            return m_age;  
        }  
    }  
}  
  
public static void Main (string[]  
args)  
{  
    Person pers = new Person();  
    pers.age = 20;  
    Console.WriteLine (pers.age);  
}
```

# Пример

```
enum Sex {  
    Male,  
    Female  
}  
  
private string m_name;  
private Sex m_sex;
```

```
public string name {  
    get {  
        return m_name;  
    }  
    set {  
        if (m_sex == Sex.Male)  
            m_name = "Mr. " + value;  
        else  
            m_name = "Ms. " + value;  
    }  
}
```

# Еще пример

```
public Sex sex {  
    get {  
        return m_sex;  
    }  
}
```

```
public int ageInMonth {  
    get {  
        return m_age * 12;  
    }  
}
```

# Преимущества и недостатки свойств

- Более компактный и наглядный синтаксис
- Запись в свойство не может вернуть ошибку, а только выбросить исключение
- Чтение и запись в свойство вызывает метод, что не очевидно при использовании



# Инкапсуляция

- Все данные методов должны быть приватными
- Для доступа к данным должны использоваться методы и свойства

# Конструкторы объектов

```
Student stud = new Student();
```

- Конструктор - это метод класса
- Конструктор необходим для создания объекта и инициализации данных объекта
- Каждый класс имеет как минимум один конструктор
- Конструкторов может быть несколько

# Конструктор по умолчанию

- Создается компилятором автоматически
- Не делает совершенно ничего

```
class Student {  
    public Student() {  
    }  
}
```

# Создаем свой конструктор

```
class Student {  
    public Student() {  
        Console.WriteLine("Создан новый объект");  
    }  
}
```

# Создаем полезный конструктор

```
class Student {  
    string Name;  
    int Age;  
    public Student(string n, int a) {  
        Name = n;  
        if(a > 10 && a < 100) {    Age = a;    }  
        else { a = 18; }  
    }  
}
```

```
Student stud = new Student ("Виктор", 20);
```

```
Student stud2 = new Student ();
```

# Перегрузка конструкторов

```
class Date {  
    public Date(int year, int month, int day) {  
    }  
    public Date(int year, int dayOfYear){  
    }  
    public Date(string dateInDDMMYYYYY){  
    }  
};
```

# Обработка некорректных значений в конструкторе

- Замена значениями по умолчанию
- Выбрасывание исключений

# Заполняем значением по умолчанию

```
class Circle {  
    double R;  
    public Circle(double r){  
        if(r <= 0)  
            R = 1;  
        else R = r;  
    }  
}
```



# Выбрасываем исключение

```
class Circle {  
    double R;  
    public Circle(double r){  
        if(r <= 0)  
            throw new Exception("Радиус меньше нуля!");  
        else R = r;  
    }  
}
```

# Методы (функции классов)

```
class Circle {  
    double R;  
    public Circle(double r){  
        R = r;  
    }  
    public double Area() {  
        return Math.PI * R * R;  
    }  
}
```