# Реализация принципов ООП в .NET

- Конструкторы
- Классическое наследование
- Виртуальные методы
- Абстрактные классы, абстрактные методы
- Интерфейсы

# Конструкторы

**Конструктор – это специальный метод класса, который позволяет создавать объекты данного класса, настраивая при этом их исходное состояние.**

# Конструктор

```
class Car
{
    private string _name; //название автомобиля
    private double _speed; //скорость

    //конструктор по умолчанию
    public Car()
    {
    }

    //конструкторы, принимающие параметры
    public Car(string name)
    {
        _name = name;
    }

    public Car(string name, double speed)
    {
        _name = name;
        _speed = speed;
    }
}
```

# Конструкторы

```
class Bachelor : Student //класс бакалавров
{
    private string _surname;
    private string _name;

    public Bachelor(string surname, string name)
    {
        _surname = surname;
        _name = name;
    }
}
```

# Конструкторы

```
class Master  : Student //класс магистров
{
    private string _surname;
    private string _name;

    public Master(string surname, string name)
    {
        _surname = surname;
        _name = name;
    }
}
```

# Конструктор

```
//класс, определяющий всех студентов
class Student
{
    private string _surname;
    private string _name;

    public Student(string surname, string name)
    {
        _surname = surname;
        _name = name;
    }
}
```

# Наследование

```
class Master : Student
{
    private string _surname;
    private string _name;
//автоматически устанавливаем значения
//для базового класса:
    public Master(string surname, string name)
        : base(surname, name)
    {
        _surname = surname;
        _name = name;
    } }
```

# Наследование

```
private float st = 150;
public  void Stipend(float proc)  //стипендия
{
   st += st * proc / 100;
}
private string FullName()   //полное имя
{
   return _surname + " " + _name;
}
public void ShowInf()    //информация о студенте
{
   Console.WriteLine(FullName() + " стипендия=" +
   st.ToString());
```

# Наследование

```
static void Main(string[] args)
{
    Master mag = new Master("Иванов", "Петр");
    mag.Stipend(50);
    mag.ShowInf();

    Bachelor bak = new Bachelor("Васильев", "Иван",
4);
    bak.Stipend(30);
    bak.ShowInf();
}
```

# Виртуальные методы

Задача: Если бакалавры не сдали сессию, предположим они должны получить минимальную стипендию (без начисления процентов). Успевающие на хорошо и отлично должны получить повышенную стипендию.

Минимальная оценка 2 – не сдали сессию, минимальная оценка 4 – хорошисты.

# Виртуальные методы

```
//определяем в родительском классе
// метод Stipend как виртуальный
public virtual void Stipend(float proc)
{
    …
}
```

# Виртуальные методы

//в производном классе его можно заместить:

```
public override void Stipend(float proc)
{
    if (_mark == 2) base.Stipend(0);
    else
    {
        if (_mark >= 4) base.Stipend(proc + 20);
        else base.Stipend(proc);
    }
```

# Виртуальные методы

**Виртуальный метод – это метод, который может быть замещен в производном классе.**

# Абстрактные классы

//Создаем объект родительского класса

```
Student   st = new Student("Козаков", "Костя");
st.ShowInf();  //магистр, бакалавр?
```

//Запрещаем создание экземпляров от класса:

```
abstract  class Student
```

# Абстрактные методы

**Абстрактные методы позволяют определить в базовом классе методы без реализации по умолчанию.**

# Абстрактные методы

## Геометрические фигуры: Родительский класс **Shape** и производные **Circle, Hexagon**.

```
abstract class Shape
{ public abstract void Draw() ; }

Circle circ = new Circle(); //окружность
circ.Draw();
Hexagon hex = new Hexagon();
//шестиугольник
hex.Draw();
```

# Абстрактные методы

//Этот вывод можно осуществить по-другому, //воспользовавшись полиморфизмом:

```
Shape[] sh = { new Hexagon(), new Circle() };

for (int i = 0; i < sh.Length; i++)
{
    sh[i].Draw();
}
```

# Интерфейсы

**Интерфейс – это набор абстрактных типов.**

Добавляем класс **Triangle()** – треугольник и создадим интерфейс **IPointy** с методом **GetPoint().**
Наследуем интерфейс IPointy в классах Triangle() и Hexagon():

class **Hexagon** : **Shape, Ipointy**
class **Triangle** : **Shape, Ipointy**

# Интерфейсы

```
//определяем метод в классе Hexagon:
public void GetPoint()
{
    Console.WriteLine("У этой фигуры 6 углов");
}
//определяем методы в классе Triangle:
public void GetPoint()
{
    Console.WriteLine("У этой фигуры 3 угла");
}
```

# Интерфейсы

```
//Добавляем вывод углов:
for (int i = 0; i < sh.Length; i++)
{
    sh[i].Draw();
    //проверяем на наличие интерфейса:
    if (sh[i]  is  IPoints)
        ((IPoints)sh[i]).GetPoint();
}
```

# Лабораторная работа 2

Создать на основе базового класса **Car** три класса-автомобиля: Тойота, Феррали, Бугатти. Добавить класс Радио, в котором два метода: **On(), Off().**

В классе **Car** должны быть обязательно (кроме других) методы: **Start(), Stop(), Speedup().**

Необходимая функциональность: завести транспортное средство, включить/выключить радио, придать автомобилю ускорение, остановить автомобиль ит.п.

Учесть, что максимальная скорость Тойоты не выше 300 км/ч.