Основные принципы объектно-ориентированного программирования (ООП)

Основные конструкции языка C#
2 часть

Переопределение методов и свойств

- позволяет в дочернем классе изменять функционал унаследованного от родительского

Для указания возможности переопределения в родительском классе используется модификатор **virtual,** а при переопределении в дочернем класса – модификатор **override**

Реализация переопределения методов

Базовый класс

```
internal class Person
{
    Ссылок: 2
    public string Name { get; set; }
    ссылка: 1
    public Person(string name)
    {
        Name = name;
    }
    Ссылок: 0
    public virtual void Print()
    {
        Console.WriteLine(Name);
    }
}
```

Унаследованный класс

```
class Employee : Person
{
    Ссылок: 2
    public string Company { get; set; }
    Ссылок: 0
    public Employee(string name, string company)
        : base(name)
    {
        Company = company;
    }
    ссылка: 1
    public override void Print()
    {
        Console.WriteLine($"{Name} работает в {Company}");
    }
}
```

1 способ

2 способ

```
public override void Print()
{
    base.Print();
    Console.WriteLine($"\nработает в {Company}");
}
```

Реализация переопределения свойств

Базовый класс

```
class Person
{
    int age = 1;
    Ссылок: 4
    public virtual int Age
    {
        get ⇒ age;
        set { if (value > 0 && value < 110) age = value; }
    }
    Ссылок: 2
    public string Name { get; set; }
    ссылка: 1
    public Person(string name)
    {
        Name = name;
    }
    Ссылок: 0
    public virtual void Print() ⇒ Console.WriteLine(Name);
}
```

Унаследованный класс

```
class Employee : Person
{
    Ссылок: 4
    public override int Age
    {
        get ⇒ base.Age;
        set { if (value > 17 && value < 110) base.Age = value; }
    }
    ссылка: 1
    public string Company { get; set; }
    Ссылок: 0
    public Employee(string name, string company)
        : base(name)
    {
        Company = company;
        base.Age = 18; // возраст для работников по умолчанию
    }
}
```

Запрет на переопределение

sealed применяется в паре с override только в переопределяемых методах.

```
class Employee : Person
{
    Ссылок: 2
    public string Company { get; set; }
    Ссылок: 0
    public Employee(string name, string company)
        : base(name)
    {
        Company = company;
    }
    Ссылок: 2
    public override sealed void Print()
    {
        base.Print();
        Console.WriteLine($"\nработает в {Company}");
    }
}
```

Абстрактные классы

Содержат лишь самую общую форму реализации для всех его производных классов, а наполнение ее деталями предоставляется каждому из этих классов.

- Создавать экземпляры абстрактного класса нельзя
- Абстрактные классы могут определять абстрактные методы
- Абстрактные методы не имеют реализации, поэтому определение такого метода заканчивается точкой с запятой
- Абстрактные члены классов не должны иметь модификатор private.
- Производный класс обязан реализовать все абстрактные члены базового класса. При отказе от реализации производный класс также должен быть определен как абстрактный

```
abstract class UserInfo
{
    protected string Name;
    protected byte Age;

    ссылка: 1
    public UserInfo(string Name, byte Age)
    {
        this.Name = Name;
        this.Age = Age;
    }
    ссылка: 1
    public abstract string ui();
}
```

```
class UserFamily : UserInfo
{
    string Family;

    Ссылок: 0
    public UserFamily(string Family, string Name, byte Age) : base(Name, Age)
    {
        this.Family = Family;
    }
    ссылка: 1
    public override string ui()
    {
        return Family + " " + Name + " " + Age;
    }
}
```

Пример абстрактного класса фигуры

```
abstract class Shape // абстрактный класс фигуры
{
    Ссылок: 2
    public abstract double GetPerimeter();
    Ссылок: 2
    public abstract double GetArea();
}
```

```
class Rectangle : Shape  // производный класс прямоугольника
{
    Ссылок: 2
    public float Width { get; set; }
    Ссылок: 2
    public float Height { get; set; }

    Ссылок: 2
    public override double GetPerimeter() => Width * 2 + Height * 2;
    Ссылок: 2
    public override double GetArea() => Width * Height;
}
```

```
class Circle : Shape // производный класс окружности
{
    Ссылок: 3
    public double Radius { get; set; }
    Ссылок: 2
    public override double GetPerimeter() => Radius * 2 * 3.14;
    Ссылок: 2
    public override double GetArea() => Radius * Radius * 3.14;
}
```

Обобщённые/универсальные типы (generics)

- позволяют создавать классы, структуры, интерфейсы, методы и делегаты, в которых обрабатываемые данные указываются в виде параметра

Обобщенный класс – это класс, который оперирует параметризированным типом данных

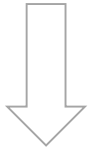
Имена обобщенных типов должны начинаться с буквы T.

Преимущества:

- Производительность
- Безопасность
- Повторное использование двоичного кода

Пример использования обобщенных типов

```
class Person
{
    ссылка: 1
    public object Id { get; }
    ссылка: 1
    public string Name { get; }
    Ссылок: 2
    public Person(object id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

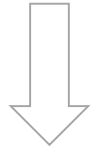


```
Person tom = new Person(546, "Tom");
Person bob = new Person("abc123", "Bob");

int tomId = (int)tom.Id;
string bobId = (string)bob.Id;
```

```
class Person
{
    ссылка: 1
    public int Id { get; }
    ссылка: 1
    public string Name { get; }
    Ссылок: 0
    public Person(int id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

```
class Person<T>
{
    ссылка: 1
    public T Id { get; set; }
    ссылка: 1
    public string Name { get; set; }
    Ссылок: 2
    public Person(T id, string name)
    {
        Id = id;
        Name = name;
    }
}
```



```
Person<int> tom = new Person<int>(546, "Tom");
Person<string> bob = new Person<string>("abc123", "Bob");

int tomId = tom.Id;
string bobId = bob.Id;
```