# Основы Бэкенда на C#

# ООП

## Наследование

Принцип парадигмы ООП, согласно которому наследники родительского класса автоматически перенимают все поля родителя и доступны при обращении именно к наследнику.

namespace OOP

{

class Program1

{

class Animal

{

private string Kingdom = "Animal";

public void What\_kingdom()

{

Console.WriteLine(Kingdom);

}

}

class Bird : Animal

{

}

public static void Main()

{

Bird bird = new();

bird.What\_kingdom();

}

}  
}

## Инкапсуляция

Принцип парадигмы ООП, согласно которому поля одного класса могут иметь разные уровни доступа для других классов, которые пытаются получить доступ к полям этого класса.

Public везде

Private только внутри данного класса (в наследниках, внутренних и внешних классах нельзя)

Protected доступен только в наследниках: ВО ВНЕШНИХ КЛАССОВ (program) и ЛЮБЫХ других классах эти поля не доступны.

namespace OOP

{

class Program1

{

class Animal

{

private string Kingdom = "Animal";

private/protected void What\_kingdom()

{

Console.WriteLine(Kingdom);

}

}

class Bird : Animal

{

}

public static void Main()

{

Bird bird = new();

bird.What\_kingdom(); - ошибка

}

}  
}

## Полиморфизм

Принцип парадигмы ООП, согласно которому наследники родительского класса могут переопределять методы реализованные у родительского класса, помеченные как виртуальные.

namespace OOP

{

class Program1

{

class Animal

{

private string Kingdom = "Animal";

public virtual void What\_kingdom()

{

Console.WriteLine(Kingdom);

}

}

class Bird : Animal

{

public override void make\_sound()

{

Console.WriteLine("tweet tweet");

}

}

public static void Main()

{

Animal animal = new();

Bird bird = new();

animal.make\_sound();

bird.make\_sound();

}

}  
}

## Абстракция

Принцип парадигмы ООП, согласно которому несколько классов, реализующих решение каких -то частных проблем, должны обобщаться в более абстрактные структуры, характеризующие их всех одновременно. И такие абстрактные структуры не должны скрывать реализацию, а предоставлять только интерфейс, с которым пользователь

# Solid

single responsibility principle

open-closed principle

Liskov substitution principle

interface segregation principle

dependency inversion principle.