**11 ПОЛИМОРФИЗМ**

Задание №1. Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с указанием названий и полей, и их значений соответствующего объекта. Составить тестирующую программу с выдачей протокола на экран монитора. При этом создать объекты базового и производных типов, используя полиморфный контейнер - массив ссылок базового класса на объекты базового и производных классов (количество объектов >=5). Родительский класс: птица (поле – название породы). Потомки: аист (поле – размах крыльев L в см), ворона (поле L – высота в см). Полиморфные методы: количество пищи в день аист – L\*1/2000, ворона – 0,8 \* L.

Листинг программы:

try

{

var s = new List<Bird>(){new Bird("Unidentified Bird"),

new Stork("Stork",5), new Crow("Crow",5),

new Stork("Stork",6), new Crow("Crow",6),

new Stork("Stork",7), new Crow("Crow",7)};

foreach (Bird p in s)

{

p.Calculate();

Console.WriteLine();

}

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

class Bird

{

private string \_Breed { get; set; }

public Bird(string breed)

{

this.\_Breed = breed;

}

public virtual void Calculate() //virtual - можем переписывать этот метод (метод Calculate())

{

Console.WriteLine("Calculating the {0} / Object type is: {1}", this.\_Breed, this.GetType());

}

}

class Stork : Bird

{

private double wingspanL;

public Stork(string breed, double wingspanL) : base(breed)

{

this.wingspanL = wingspanL;

}

public override void Calculate()

{

base.Calculate(); //base - вызывает этот метод в родительском классе (метод Calculate())

Console.WriteLine($"Wingspan = {wingspanL}");

Console.WriteLine($"Result = {(double)(wingspanL \* 1 / 2000)}");

}

}

class Crow : Bird

{

private double heightL;

public Crow(string breed, double heightL) : base(breed)

{

this.heightL = heightL;

}

public override void Calculate() // override - используется, чтобы переписать метод (метод Calculate() родительского класса)

{

base.Calculate();

Console.WriteLine($"height = {heightL}");

Console.WriteLine($"Result = {(double)(0.8 \* heightL)}");

}

}

Таблица 11.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Calculating the Unidentified Bird / Object type is: Bird  Calculating the Stork / Object type is: Stork  Wingspan = 5  Result = 0,0025  Calculating the Crow / Object type is: Crow  height = 5  Result = 4  Calculating the Stork / Object type is: Stork  Wingspan = 6  Result = 0,003  Calculating the Crow / Object type is: Crow  height = 6  Result = 4,800000000000001  Calculating the Stork / Object type is: Stork  Wingspan = 7  Result = 0,0035  Calculating the Crow / Object type is: Crow  height = 7  Result = 5,6000000000000005 |

Анализ результатов:

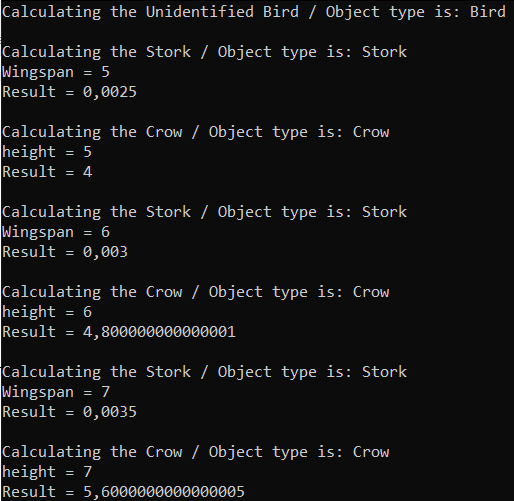


Рисунок 11.1 – Результаты работы программы

Источник: собственная разработка

Задание №2. Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 2. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов >=5). Найти количество пищи, необходимой аистам и количество пищи, необходимой воронам.

Листинг программы:

try

{

var s = new List<Bird>(){new Stork("Stork",5), new Crow("Crow",5), new Stork("Stork", 6), new Crow("Crow", 6), new Stork("Stork", 7), new Crow("Crow", 7) };

foreach (Bird p in s)

{

p.Calculate();

Console.WriteLine();

}

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

abstract class Bird

{

public abstract string Breed { get; set; }

public abstract void Calculate();

}

class Stork : Bird

{

public override string Breed { get ; set; }

private double wingspanL;

public Stork(string breed, double wingspanL)

{

this.Breed = breed;

this.wingspanL = wingspanL;

}

public override void Calculate()

{

Console.WriteLine("Calculating the {0} / Object type is: {1}", this.Breed, this.GetType());

Console.WriteLine($"Wingspan = {wingspanL}");

Console.WriteLine($"Result = {(double)(wingspanL \* 1 / 2000)}");

}

}

class Crow : Bird

{

public override string Breed { get; set; }

private double heightL;

public Crow(string breed, double heightL)

{

this.Breed = breed;

this.heightL = heightL;

}

public override void Calculate() // override - используется, чтобы переписать метод (метод Calculate() родительского класса)

{

Console.WriteLine("Calculating the {0} / Object type is: {1}", this.Breed, this.GetType());

Console.WriteLine($"height = {heightL}");

Console.WriteLine($"Result = {(double)(0.8 \* heightL)}");

}

}

Таблица 11.2 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Calculating the Stork / Object type is: Stork  Wingspan = 5  Result = 0,0025  Calculating the Crow / Object type is: Crow  height = 5  Result = 4  Calculating the Stork / Object type is: Stork  Wingspan = 6  Result = 0,003  Calculating the Crow / Object type is: Crow  height = 6  Result = 4,800000000000001  Calculating the Stork / Object type is: Stork  Wingspan = 7  Result = 0,0035  Calculating the Crow / Object type is: Crow  height = 7  Result = 5,6000000000000005 |

Анализ результатов:

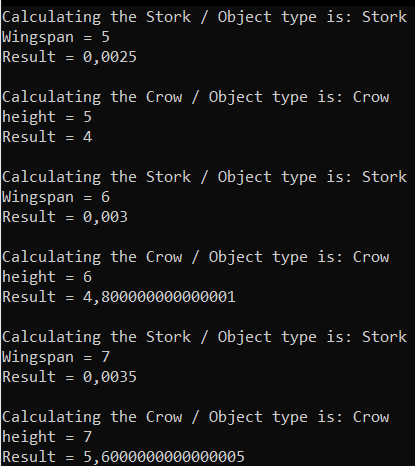


Рисунок 11.2 – Результаты работы программы

Источник: собственная разработка