Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Polymorphism

Submitted By:
Peter Stacey
pstacey
2020/04/28 08:48

Tutor:
Dipto Pratyaksa

Outcome	**Weight**
Evaluate Code	♦♦◊◊◊
Principles	♦♦♦♦◊
Build Programs	♦♦◊◊◊
Design	♦◊◊◊◊
Justify	♦♦♦◊◊

The task explores polymorphism, which is one of the code principles of OOP and directly related to the principles outcome. There is no code design in this task, however there is evaluation of code and understanding what the output will be.The task implements a small set of classes, which aligns with the code outcome and we develop a UML diagram, which relates both the design and to evaluating code and representing it using standard conventions.

April 28, 2020

Produced by Doubtfire

```cs
using System;

namespace Task_6_1P
{
    /// <summary>
    /// Prototype for a Bird and base class for more specific bird classes
    /// </summary>
    class Bird
    {
        // Instance variables
        public string Name { get; set; }

        /// <summary>
        /// Creates a new Bird
        /// </summary>
        public Bird()
        {

        }

        /// <summary>
        /// Allows the bird to fly
        /// </summary>
        public virtual void fly()
        {
            Console.WriteLine("Flap, Flap, Flap");
        }

        /// <summary>
        /// Returns a string representation of a bird
        /// </summary>
        /// <returns>
        /// String representation of a Bird
        /// </returns>
        public override string ToString()
        {
            return "A bird named " + Name;
        }
    }
}
```

```cs
using System;

namespace Task_6_1P
{
    /// <summary>
    /// Prototype for a Penguin as a type of Bird
    /// </summary>
    class Penguin : Bird
    {
        /// <summary>
        /// Prevents a Penguin from flying
        /// </summary>
        public override void fly()
        {
            Console.WriteLine("Penguins cannot fly");
        }

        /// <summary>
        /// Returns a string representation of a Penguin
        /// </summary>
        /// <returns>
        /// String representation of a Penguin
        /// </returns>
        public override string ToString()
        {
            return "A penguin named " + Name;
        }
    }
}
```

```cs
using System;

namespace Task_6_1P
{
    /// <summary>
    /// Prototype for a Duck as a type of Bird
    /// </summary>
    class Duck : Bird
    {
        // Instance variables
        public double Size { get; set; }
        public string Kind { get; set; }

        /// <summary>
        /// Returns a string representation of a Duck
        /// </summary>
        /// <returns>
        /// String representation of a Duck
        /// </returns>
        public override string ToString()
        {
            return "A duck named " + Name + " is a " + Size + " inch " + Kind;
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace Task_6_1P
{
    class Program
    {
        static void Main(string[] args)
        {
            // Step 2 - Part 1
            Bird bird1 = new Bird();
            Bird bird2 = new Bird();

            bird1.Name = "Feathers";
            bird2.Name = "Polly";

            Console.WriteLine(bird1.ToString());
            bird1.fly();

            Console.WriteLine(bird2.ToString());
            bird2.fly();

            // Step 2 - Part 2
            Penguin penguin1 = new Penguin();
            Penguin penguin2 = new Penguin();

            penguin1.Name = "Happy Feet";
            penguin2.Name = "Gloria";

            Console.WriteLine(penguin1.ToString());
            penguin1.fly();

            Console.WriteLine(penguin2.ToString());
            penguin2.fly();

            Duck duck1 = new Duck();
            Duck duck2 = new Duck();

            duck1.Name = "Daffy";
            duck1.Size = 15;
            duck1.Kind = "Mallard";

            duck2.Name = "Donald";
            duck2.Size = 20;
            duck2.Kind = "Decoy";

            Console.WriteLine(duck1.ToString());
            Console.WriteLine(duck2.ToString());

            // Step 2 - Part 3
            List<Bird> birds = new List<Bird>();
            Bird bird3 = new Bird();
            bird3.Name = "Feathers";
```

```
54              Bird bird4 = new Bird();
55              bird4.Name = "Polly";
56
57              Penguin penguin3 = new Penguin();
58              penguin3.Name = "Happy Feet";
59              Penguin penguin4 = new Penguin();
60              penguin4.Name = "Gloria";
61
62              Duck duck3 = new Duck();
63              duck3.Name = "Daffy";
64              duck3.Size = 15;
65              duck3.Kind = "Mallard";
66
67              Duck duck4 = new Duck();
68              duck4.Name = "Donald";
69              duck4.Size = 20;
70              duck4.Kind = "Decoy";
71
72              birds.Add(bird3);
73              birds.Add(bird4);
74              birds.Add(penguin3);
75              birds.Add(penguin4);
76              birds.Add(duck3);
77              birds.Add(duck4);
78
79              birds.Add(new Bird { Name = "Birdy" });
80
81              foreach (Bird bird in birds)
82              {
83                  Console.WriteLine(bird);
84              }
85
86              // Part 3
87              Duck duck5 = new Duck();
88              duck5.Name = "Daffy";
89              duck5.Size = 15;
90              duck5.Kind = "Mallard";
91
92              Duck duck6 = new Duck();
93              duck6.Name = "Donald";
94              duck6.Size = 20;
95              duck6.Kind = "Decoy";
96
97              List<Duck> ducksToAdd = new List<Duck>()
98              {
99                  duck5,
100                 duck6
101             };
102
103             IEnumerable<Bird> upcastDucks = ducksToAdd;
104
105             List<Bird> birds2 = new List<Bird>();
106             birds2.Add(new Bird() { Name = "Feather" });
```

```
107
108             birds2.AddRange(upcastDucks);
109
110             foreach (Bird bird in birds2)
111             {
112                 Console.WriteLine(bird);
113             }
114         }
115     }
116 }
```

SIT232 – Object Oriented Development Task 6.1P- Report on Polymorphism

Student Name: Peter Stacey

Student ID: 219011171

UML Diagram

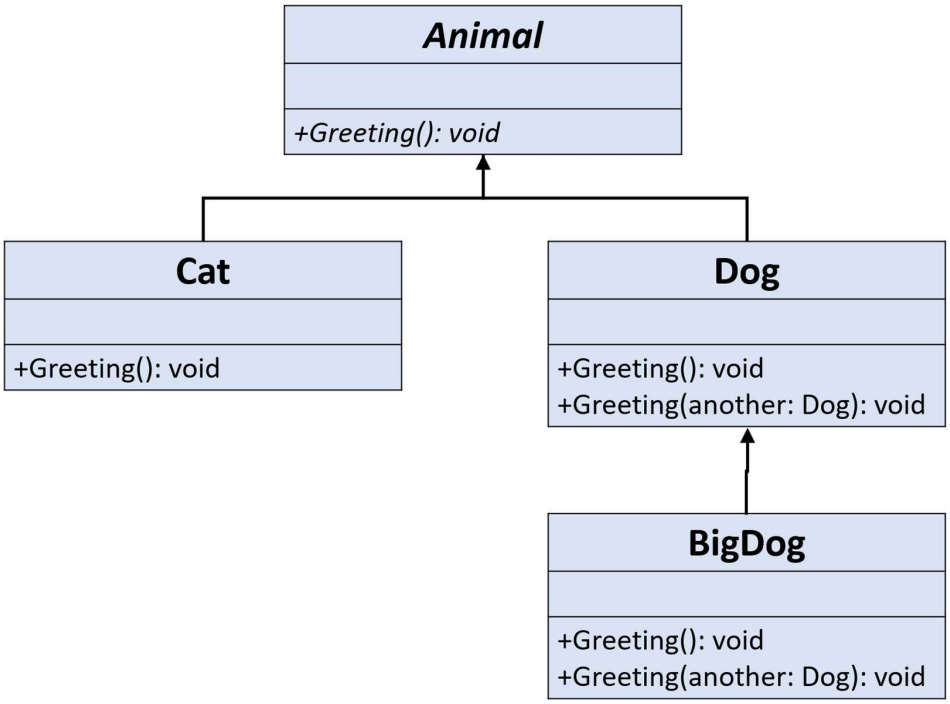

Figure 1: UML Diagram of Inheritance Hierarchy

Explanation of outputs

Ignoring the syntax error, with the classes defining methods Greeting (capitalized) and the TestAnimal class calling greeting (all lower-case), the following outputs and or errors occur:

Lines	Output/Error
`Cat cat1 = new Cat();` `cat1.Greeting();`	Cat: meow
`Dog dog1 = new Dog();` `dog1.Greeting();`	Dog: woof
`BigDog bigDog1 = new BigDog();` `bigDog1.Greeting();`	BigDog: Woow
`Animal animal1 = new Cat();` `animal1.Greeting();`	Cat: meow
`Animal animal2 = new Dog();` `animal2.Greeting();`	Dog: woof
`Animal animal3 = new BigDog();` `animal3.Greeting();`	BigDog: Woow
`Animal animal4 = new Animal();`	Error: Abstract classes cannot be instantiated

`Dog dog2 = (Dog)animal2;`
`BigDog bigDog2 = (BigDog)animal3;`
`Dog dog3 = (Dog)animal3;`
	No output from these lines, but also no problems. These are all valid casts.

`Cat cat2 = (Cat)animal2;`	Error: This is an invalid cast, as animal2 has already been cast to a Dog, which is not in the same branch of the inheritance tree.

`dog2.Greeting(dog3);`
`dog3.Greeting(dog2);`
`dog2.Greeting(bigDog2);`
`bigDog2.Greeting(dog2);`
`bigDog2.Greeting(bigDog1);`
	Dog: wooooooffffff
	Dog: wooooooffffff
	Dog: wooooooffffff
	Wooooooooowwwww
	Wooooooooowwwww
	
	In all cases, the calls are to the overloaded Greeting that accepts another Dog as a parameter, so the output matches the body of those methods.