

ASSIGNMENT 1

Qualification	BTEC Level 5 HND Diploma in Computing		
Unit number and title	Unit 20: Advanced Programming: Examine and design solutions with OOP and Design Patterns		
Submission date		Date Received 1st submission	
Re-submission Date		Date Received 2nd submission	
Student Name	Nguyen Phuc Son Tung	Student ID	GCH16064
Student Name	Nguyen Cong Nhat	Student ID	GCD17202
Class	GCD0604	Assessor name	Hoang Nhu Vinh
<p align="center">Student declaration</p> <p align="center">I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.</p>			
		Student's signature	Nguyen Phuc Son Tung
		Student's signature	Nguyen Cong Nhat

Grading grid

P1	P2	M1	M2	D1	D2

☐ **Summative Feedback:**☐ **Resubmission Feedback:****Grade:****Assessor Signature:****Date:****Internal Verifier's Comments:****Signature & Date:**

Contents

INTRODUCTION	3
BODY	4
P1 + P2 Examine the characteristics of the object-orientated paradigm as well as the various class relationships and build class diagrams using a UML tool.....	4
Class & Object	4
Abstraction	6
Inheritance.....	10
Encapsulation	14
Polymorphism	17
CONCLUSION.....	20
REFERENCES	21
Figure 1: Class diagram of class & object	4
Figure 2: Result of class & object.....	5
Figure 3: Class diagram of abstraction	6
Figure 4: Result of abstraction	9
Figure 5: Class diagram of inheritance	11
Figure 6: Result of inheritance.....	13
Figure 7: Class diagram of encapsulation	15
Figure 8: Result of encapsulation.....	16
Figure 9: Class diagram of polymorphism.....	17
Figure 10: Result of polymorphism	19

INTRODUCTION

We have recently joined a software development company to help improve their documentation of their in-house software libraries which were developed with very poor documentation. As a result, it has been very difficult for the company to utilize their code in multiple projects due to poor documentation. Our role is to alleviate this situation by showing the efficient of UML diagrams in OOAD and Design Patterns in usages.

In first task, we and our team will explain characteristics of Object-oriented programming paradigm by applying Object-oriented analysis and design on a given (assumed) scenario. The scenario can be small but should be able to presents various characteristics of OOP (such as: encapsulation, inheritance, polymorphism, override, overload, etc.).

The second task is to introduce some design patterns (including 3 types: creational, structural and behavioral) to audience by giving real case scenarios, corresponding patterns illustrated by UML class diagrams.

To summarize, we will analyze the relationship between the object-orientated paradigm and design patterns.

BODY

P1 + P2 Examine the characteristics of the object-orientated paradigm as well as the various class relationships and build class diagrams using a UML tool.

Class & Object

We implement Class Animals in C # with fields: name, age, gender and the Get_infor () method.

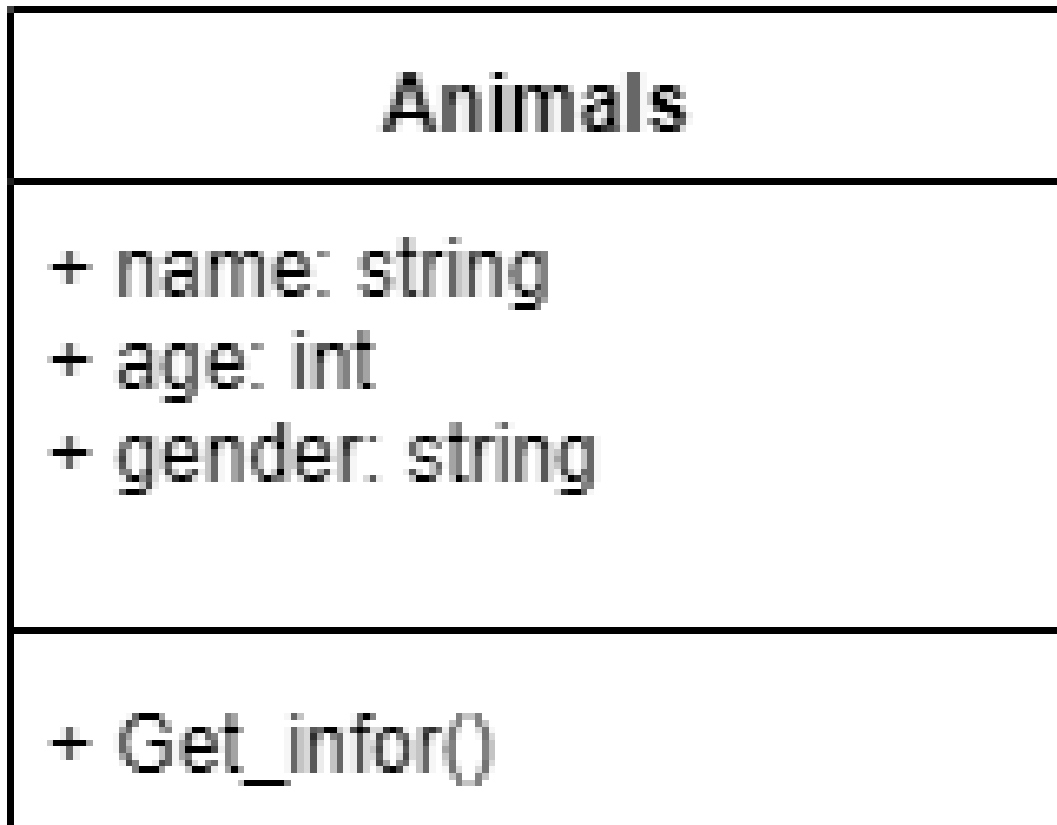


Figure 1: Class diagram of class & object

Implement program in C#

```
class Animals
{
    public string name;
    public int age;
    public string gender;
    public void setname(string Name)
    {
        name = Name;
    }
    public void setage(int Age)
    {
        age = Age;
    }
    public void setgender(string Gender)
    {
        gender = Gender;
    }
    public void Get_infor()
    {
        Console.WriteLine($"Name: {name} is {age} day -- Gender: {gender}");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Animals myanimals = new Animals();
        myanimals.setname("black Dog");
        myanimals.setage(4);
        myanimals.setgender("f");
        myanimals.Get_infor();
        Console.ReadKey();
    }
}
```

Result

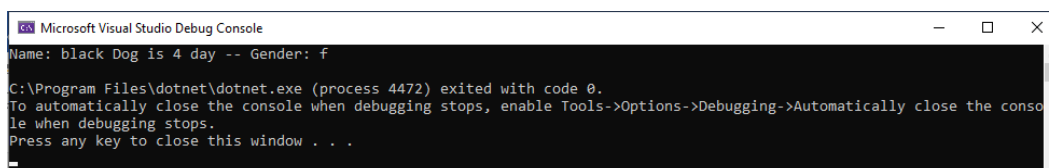


Figure 2: Result of class & object

Abstraction

Definition of Abstraction

In software engineering and computer science, **abstraction** is the process of removing physical, spatial, or temporal details or attributes in the study of objects or systems in order to focus attention on details of higher importance, it is also very similar in nature to the process of generalization. The creation of abstract concept-objects which are created by mirroring common features or attributes from various non-abstract objects or systems of study - the result of the process of abstraction according (Wikipedia, Abstraction (computer science), 2019).

Class Diagram of Abstraction

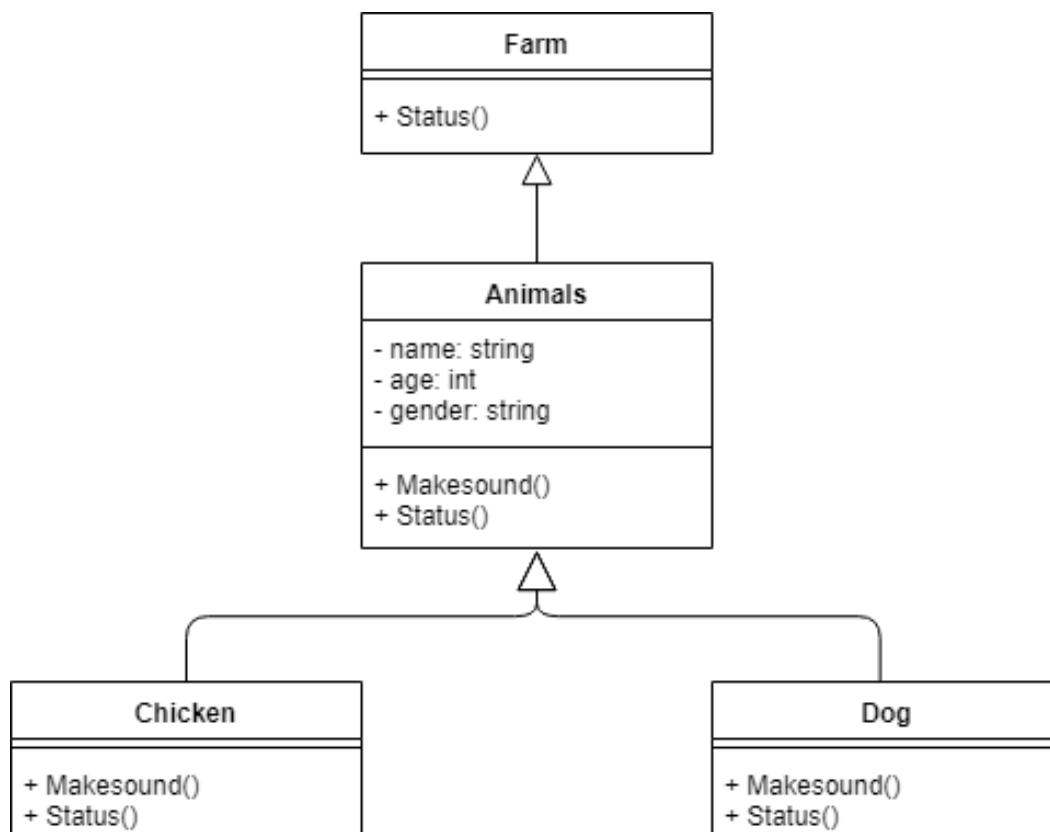


Figure 3: Class diagram of abstraction

Implement program in C#

```
public abstract class Farm
{
    public virtual void Status()
    { }
}
public class Animal : Farm
{
    private string name;
    private int age;
    private string gender;
    protected Animal(string name, int age, string gender)
    {
        this.Name = name;
        this.Age = age;
        this.Gender = gender;
    }
    public string Name
    {
        get{return this.name;}
        set{this.name = value;}
    }
    public int Age
    {
        get{return this.age;}
        set{this.age = value;}
    }
    public string Gender
    {
        get{return this.gender;}
        set{this.gender = value;}
    }
    public virtual void MakeSound()
    {
        Console.WriteLine("empty");
    }
    public override void Status()
    {
        Console.WriteLine("==Information of Animal==");
    }
}
```



```

public class Chicken : Animal
{
    public Chicken(string name, int age, string gender) : base(name, age, gender) { }
    public override void MakeSound()
    {
        Console.WriteLine("O o O o");
    }
    public override void Status()
    {
        Console.WriteLine("== Information of Chicken ==");
        Console.WriteLine($"Name: {Name}, Age: {Age}, Gender: {Gender}");
        Console.WriteLine("=====");
    }
}

public class Dog : Animal
{
    public Dog(string name, int age, string gender)
        : base(name, age, gender)
    { }
    public override void MakeSound()
    {
        Console.WriteLine("Gau gau Gau gau");
    }
    public override void Status()
    {
        Console.WriteLine("== Information of Dog ==");
        Console.WriteLine($"Name: {Name}, Age: {Age}, Gender: {Gender}");
        Console.WriteLine("=====");
    }
}

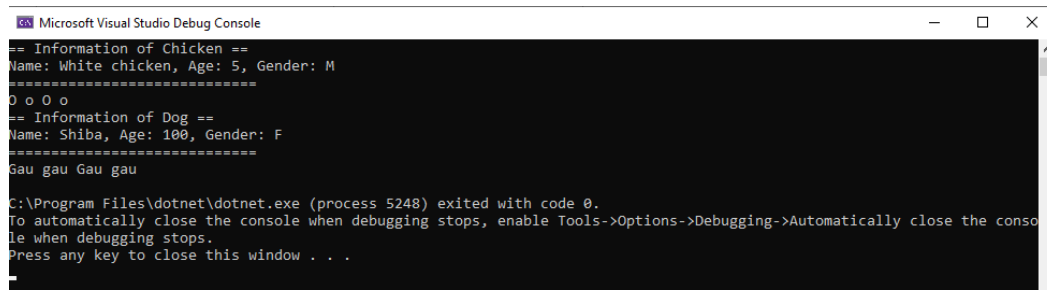
```

```

class Program
{
    static void Main(string[] args)
    {
        Chicken mychicken = new Chicken("White chicken", 5, "M");
        Dog mydog = new Dog("Shiba", 100, "F");
        mychicken.Status();
        mychicken.MakeSound();
        mydog.Status();
        mydog.MakeSound();
        Console.ReadKey();
    }
}

```

Result



```
Microsoft Visual Studio Debug Console
== Information of Chicken ==
Name: White chicken, Age: 5, Gender: M
=====
O o O o
== Information of Dog ==
Name: Shiba, Age: 100, Gender: F
=====
Gau gau Gau gau
C:\Program Files\dotnet\dotnet.exe (process 5248) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 4: Result of abstraction

The Abstract class Farm is created with the Status() method but it is empty, then with the Inheritance classes it will show how the Status() method works for each class.

Inheritance

Definition of Inheritance

In object-oriented programming, according (Wikipedia, Inheritance (object oriented programming), 2019), *inheritance* is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation. Also defined as deriving new classes (sub classes) from existing ones (super class or base class) and forming them into a hierarchy of classes. In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object (except: constructors, destructor, overloaded operators and friend functions of the base class). Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a directed graph.

Class Diagram of Inheritance

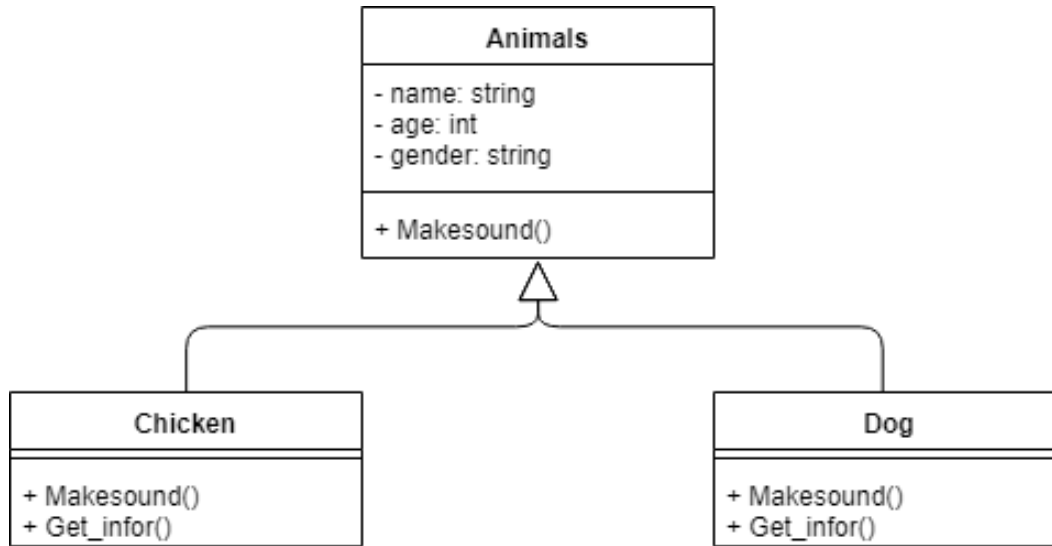


Figure 5: Class diagram of inheritance

Chicken class and Dog Inheritance class Animals class, 2 sub classes do not need to initialize fields, they base fields already exist in Animals class.

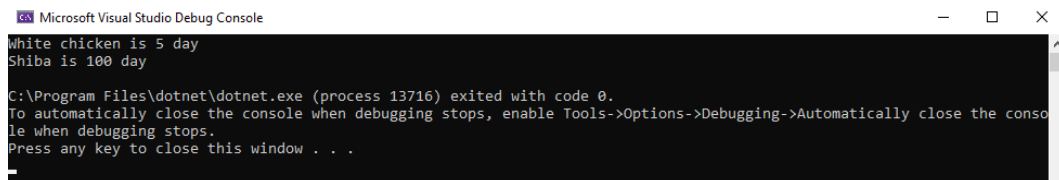
Implement program in C#

```
public abstract class Animals
{
    private string name;
    private int age;
    private string gender;
    protected Animals(string name, int age, string gender)
    {
        this.Name = name;
        this.Age = age;
        this.Gender = gender;
    }
    public string Name
    {
        get{return this.name;}
        set{this.name = value;}
    }
    public int Age
    {
        get{return this.age;}
        set{this.age = value;}
    }
    public string Gender
    {
        get{return this.gender;}
        set{this.gender = value;}
    }
    public virtual string MakeSound()
    {
        return string.Empty;
    }
}
```

```
public class Chicken : Animals
{
    public Chicken(string name, int age, string gender) : base(name, age, gender) { }
    public override string MakeSound() {
        return "O o O o";
    }
    public void Getinfor() {
        Console.WriteLine($"{Name} is {Age} day");
    }
}
public class Dog : Animals
{
    public Dog(string name, int age, string gender)
        : base(name, age, gender) { }
    public override string MakeSound()
    {
        return "BauBau";
    }
    public void Getinfor()
    {
        Console.WriteLine($"{Name} is {Age} day");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Chicken mychicken = new Chicken("White chicken", 5, "M");
        Dog mydog = new Dog("Shiba", 100, "F");
        mychicken.Getinfor();
        mydog.Getinfor();
        Console.ReadKey();
    }
}
```

Result



```
Microsoft Visual Studio Debug Console
white chicken is 5 day
Shiba is 100 day
C:\Program Files\dotnet\dotnet.exe (process 13716) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 6: Result of inheritance

Encapsulation

Definition of Encapsulation

According (Wikipedia, Encapsulation (computer programming), 2019), *encapsulation* in object-oriented programming, is the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called getters and setters) to access the values, and other client classes call these methods to retrieve and modify the values within the object.

Characteristics of Encapsulation

- ✓ Encapsulation protects an object from unwanted access by clients.
- ✓ Encapsulation allows access to a level without revealing the complex details below that level.
- ✓ It reduces human errors.
- ✓ Simplifies the maintenance of the application
- ✓ Makes the application easier to understand.
- ✓ The main advantage of using encapsulation is the security of the data.
- ✓ Everyone knows how to access it.
- ✓ Can be easily used regardless of implementation details.
- ✓ There should not be any side effects of the code for the rest of the application.
- ✓ The idea of packaging is to keep the layers separate and prevent them from closely interlinking.

Class Diagram of Encapsulation

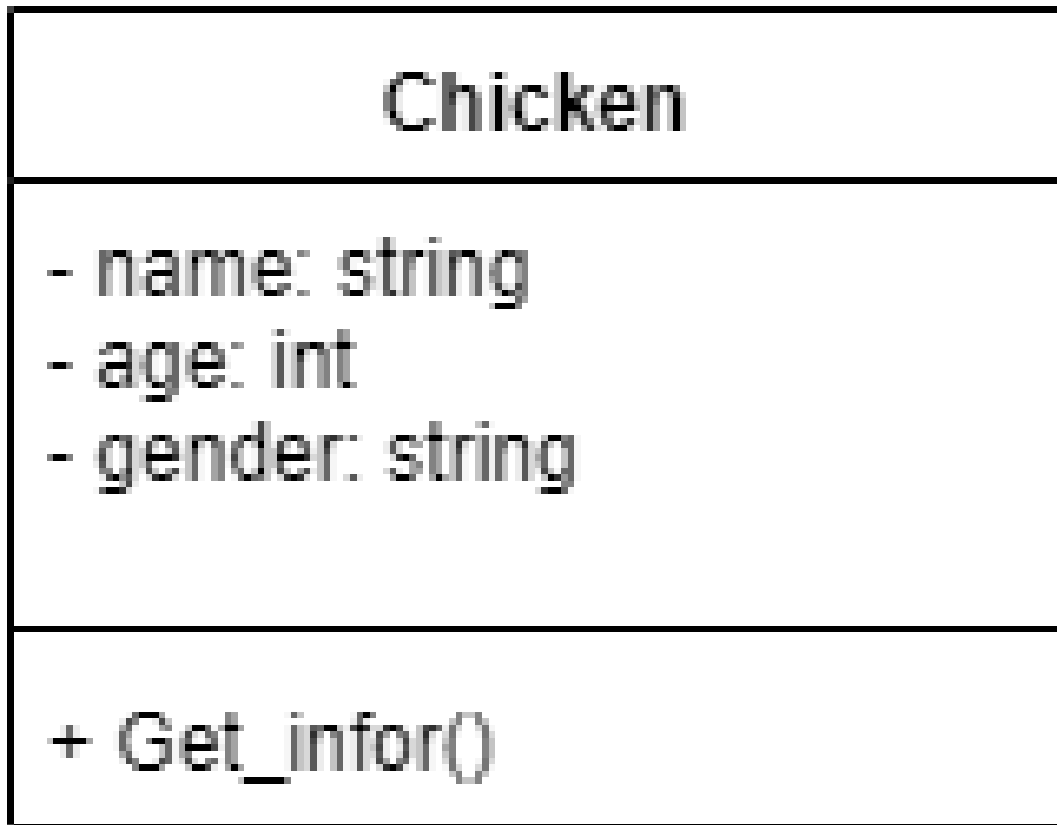


Figure 7: Class diagram of encapsulation

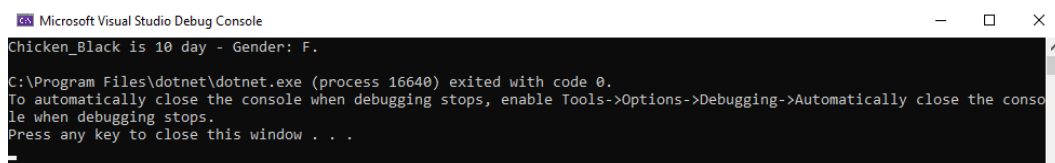
In the Chicken class, there are private fields, I initialized them methods() to get and set, when calling the fields in the Chicken class, we cannot query it but only query to the method () set and get.

Implement program in C#

```
public class Chicken
{
    private string name;
    private int age;
    private string gender;
    public string Name {
        get{return this.name;}
        set{this.name = value;}
    }
    public int Age {
        get{return this.age;}
        set{this.age = value;}
    }
    public string Gender {
        get { return this.gender; }
        set { this.gender = value; }
    }
    public void Get_infor() {
        Console.WriteLine($"{Name} is {Age} day - Gender: {Gender}.");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Chicken mychicken = new Chicken();
        mychicken.Name = "Chicken_Black";
        mychicken.Age = 10;
        mychicken.Gender = "F";
        mychicken.Get_infor();
        Console.ReadKey();
    }
}
```

Result



```
Microsoft Visual Studio Debug Console
Chicken_Black is 10 day - Gender: F.
C:\Program Files\dotnet\dotnet.exe (process 16640) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 8: Result of encapsulation

Polymorphism

Definition of Polymorphism

In programming languages and type theory, **polymorphism** is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types according (Wikipedia, Polymorphism (computer science), 2019).

The most commonly recognised major classes of polymorphism are:

- ✓ Ad hoc polymorphism: defines a common interface for an arbitrary set of individually specified types.
- ✓ Parametric polymorphism: when one or more types are not specified by name but by abstract symbols that can represent any type.
- ✓ Subtyping (also called subtype polymorphism or inclusion polymorphism): when a name denotes instances of many different classes related by some common superclass.

Class Diagram of Polymorphism

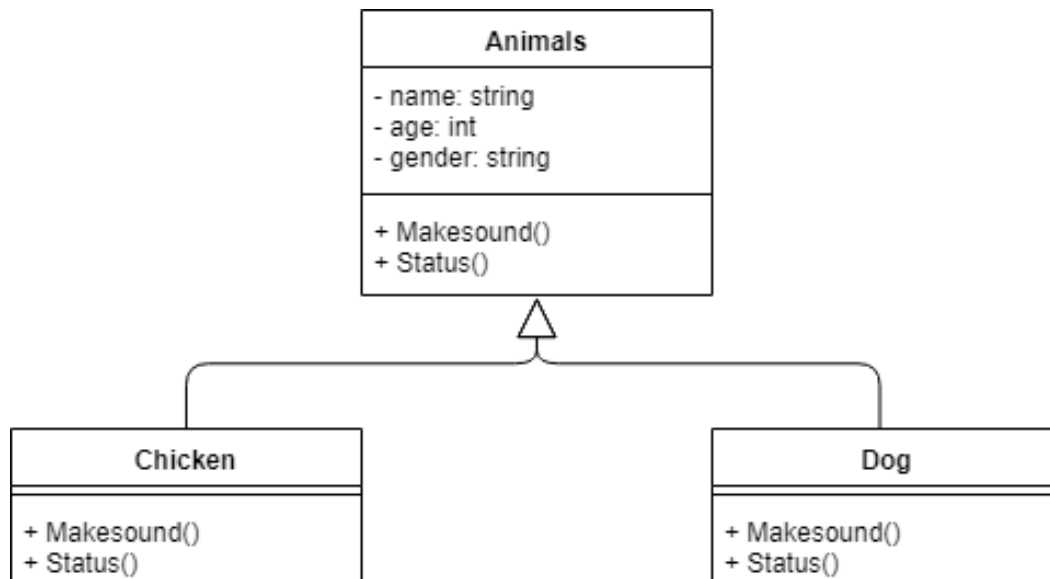


Figure 9: Class diagram of polymorphism

Implement program in C#

```
public class Animal
{
    private string name;
    private int age;
    private string gender;
    protected Animal(string name, int age, string gender)
    {
        this.Name = name;
        this.Age = age;
        this.Gender = gender;
    }
    public string Name
    {
        get {
            return this.name;
        }
        set {
            this.name = value;
        }
    }
    public int Age
    {
        get {
            return this.age;
        }
        set {
            this.age = value;
        }
    }
    public string Gender
    {
        get {
            return this.gender;
        }
        set {
            this.gender = value;
        }
    }
    public virtual void MakeSound()
    {
        Console.WriteLine("Sound of animals");
    }
    public virtual void Status()
    {
        Console.WriteLine("== Information of Animal ==");
    }
}
```

```
public class Chicken : Animal {
    public Chicken(string name, int age, string gender) : base(name, age, gender) { }
    public override void MakeSound() { Console.WriteLine("O o O o"); }
    public override void Status()
    {
        Console.WriteLine("== Information of Chicken ==");
        Console.WriteLine($"Name: {Name}, Age: {Age}, Gender: {Gender}");
        Console.WriteLine("=====");
    }
}
```

```
public class Dog : Animal {
    public Dog(string name, int age, string gender)
        : base(name, age, gender) { }
    public override void MakeSound()
    { Console.WriteLine("Gau gau Gau gau"); }
    public override void Status()
    {
        Console.WriteLine("== Information of Dog ==");
        Console.WriteLine($"Name: {Name}, Age: {Age}, Gender: {Gender}");
        Console.WriteLine("=====");
    }
}
```

```
class Program {
    static void Main(string[] args) {
        Chicken mychicken = new Chicken("White chicken", 5, "M");
        Dog mydog = new Dog("Shiba", 100, "F");
        mychicken.Status();
        mychicken.MakeSound();
        mydog.Status();
        mydog.MakeSound();
        Console.ReadKey(); }
}
```

Result

```
Microsoft Visual Studio Debug Console
== Information of Chicken ==
Name: White chicken, Age: 5, Gender: M
O o O o
== Information of Dog ==
Name: Shiba, Age: 100, Gender: F
Gau gau Gau gau
C:\Program Files\dotnet\dotnet.exe (process 9460) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 10: Result of polymorphism

CONCLUSION

In this paper, we understood each of the properties of Object-oriented programming (Abstraction, Inheritance, Polymorphism and Encapsulation.) We then implemented OOP properties in C # and represented properties in UML class diagram.

REFERENCES

Wikipedia. (2019, 11 11). *Abstraction (computer science)*. Retrieved from [https://en.wikipedia.org:https://en.wikipedia.org/wiki/Abstraction_\(computer_science\)](https://en.wikipedia.org:https://en.wikipedia.org/wiki/Abstraction_(computer_science))

Wikipedia. (2019, 10 10). *Encapsulation (computer programming)*. Retrieved from [https://en.wikipedia.org:https://en.wikipedia.org/wiki/Encapsulation_\(computer_programming\)](https://en.wikipedia.org:https://en.wikipedia.org/wiki/Encapsulation_(computer_programming))

Wikipedia. (2019, 09 09). *Inheritance (object oriented programming)*. Retrieved from [https://en.wikipedia.org:https://en.wikipedia.org/wiki/Inheritance_\(object-oriented_programming\)](https://en.wikipedia.org:https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming))

Wikipedia. (2019, 09 22). *Polymorphism (computer science)*. Retrieved from [https://en.wikipedia.org:https://en.wikipedia.org/wiki/Polymorphism_\(computer_science\)#Implementation_aspects](https://en.wikipedia.org:https://en.wikipedia.org/wiki/Polymorphism_(computer_science)#Implementation_aspects)