**Meaning**

To understand the concept of Polymorphism, it is important to know that it is composed of two Greek words: "poly" (many) and "morph" (forms). In other words, we could say that Polymorphism is described as the quality of having many forms. In programming, polymorphism is the fourth and main principle of object-oriented programming as it allows an object or line of code to behave differently depending on the context in which it is used.

To effectively apply the use of polymorphism, the principles of abstraction, encapsulation, and inheritance need to be applied. We know that derived classes can inherit attributes and methods from the superclass. However, it should be noted that polymorphism allows derived classes to override or change the behavior of inherited methods, and this is called method overriding.

**Benefits**

One benefit of polymorphism is that method overriding enables standardization when calling methods. This means that the method name remains the same in both classes, but their behavior is different. Another benefit is code reuse, which allows developers to write code that can work with different types of objects.

**Application**

An example of using polymorphism can be illustrated as follows: Let us say we have a company with three types of salespersons: 1. Salespersons with a monthly salary, 2. Commission-based salespersons without a monthly salary, 3. Hourly-based salespersons without a monthly salary. All these salespersons have common attributes such as Name, ID, Email, and Bank Account. The standard method shared by all types of salespersons is CalculateSalary(). We know that an hourly-based salesperson's salary depends on the number of hours worked, not the number of items sold. And a commission-based salesperson does not have a fixed salary and their earnings depend on the number of items sold. If they do not sell anything, they will not earn a salary. A salesperson with a monthly salary will receive a fixed salary plus commissions based on items sold. Three workers receive money, but their calculations are different.

What we would do is create a base class that shares all the common methods and attributes with the other classes. We can do it as follows: Base Class: Employee, Derived Classes: CommissioningEmployee, HourlyEmployee, and SalariedEmployee. In the base class, we place the CalculatePayment() method, and in the derived classes, we keep the same method name CalculatePayment(), but we modify the salary calculation within the CalculatePayment() method based on the conditions of each employee type. All created objects will have a class that behaves differently depending on the context in which they are used.

**Code**

public abstract class Goal

{

    protected string \_nameGoal;

    protected string \_descriptionGoal;

    protected int \_pointsPerGoal;

    protected int \_pointsEarneds;

    protected List<string> \_levels;

    public Goal(string name, string description, int points)

    {

        \_nameGoal = name;

        \_descriptionGoal = description;

        \_pointsPerGoal = points;

        \_levels = new List<string>{

            "Beginner", "Ninja", "Profesional", "Elite"

        };

    }

    public Goal(string name, string description, int points, int pointsEarneds)

    {

        \_nameGoal = name;

        \_descriptionGoal = description;

        \_pointsPerGoal = points;

        \_pointsEarneds = pointsEarneds;

        \_levels = new List<string>{

            "Beginner", "Ninja", "Profesional", "Elite"

        };

    }

    // protected virtual void DisplayGoal()

    // {

    //     Console.WriteLine($"{\_nameGoal} {\_descriptionGoal}");

    // }

    public string GetNameGoal()

    {

        return \_nameGoal;

    }

    public string GetDescriptionGoal()

    {

        return \_descriptionGoal;

    }

    protected void SetPoints()

    {

        \_pointsEarneds += \_pointsPerGoal;

    }

    protected void SetBonusPoints(int points)

    {

        \_pointsEarneds += points;

    }

    public int GetEarnedPoints()

    {

        return \_pointsEarneds;

    }

    public int GetPointsPerGoal()

    {

        return \_pointsPerGoal;

    }

//Polimorphism in action

    public abstract void DisplayGoal();

    public abstract void RecordEvent();

    protected virtual void IsComplete()

    {

    }

}

public class EternalGoal : Goal

{

    public EternalGoal(string name, string description, int points) : base(name, description, points)

    {

    }

    public EternalGoal(string name, string description, int points, int earnedPoints) : base(name, description, points, earnedPoints)

    {

    }

    public override void RecordEvent()

    {

        base.SetPoints();

        Console.WriteLine($"Congratulations! You have earned {base.GetPointsPerGoal()}");

    }

    public override void DisplayGoal()

    {

        Console.WriteLine($"[ ] {base.GetNameGoal()} ({base.GetDescriptionGoal()})");

    }

}

public class SimpleGoal : Goal

{

    private bool \_checked;

    public SimpleGoal(string name, string description, int points, bool check) : base(name, description, points)

    {

        \_checked = check;

    }

    public SimpleGoal(string name, string description, int points, bool check, int earnPoints) : base(name, description, points, earnPoints)

    {

        \_checked = check;

    }

    public override bool GetChecked()

    {

        return \_checked;

    }

    protected override void IsComplete()

    {

        \_checked = true;

    }

    public override void DisplayGoal()

    {

        if(\_checked == true)

        {

            Console.WriteLine($"[X] {base.GetNameGoal()} ({base.GetDescriptionGoal()})");

        }

        else

        {

            Console.WriteLine($"[ ] {base.GetNameGoal()} ({base.GetDescriptionGoal()})");

        }

    }

    public override void RecordEvent()

    {

        if(\_checked != true)

        {

            base.SetPoints();

            Console.WriteLine($"Congratulations! You have earned {base.GetPointsPerGoal()}");

            IsComplete();

        }

        else

        {

            Console.WriteLine("Goal Already Completed!");

        }

    }

If you notice, there are methods that are overriding and redefining the implementation of the parent class, for example, the method RecordEvent() is very different among classes. This is Polymorphism.