**Polymorphism - Articulate**

Polymorphism is a fundamental concept in object-oriented programming that allows objects of different classes to be treated as if they belong to a common class or share a common interface. The term "polymorphism" comes from the Greek words "poly," meaning "many," and "morph," meaning "form."

There are two main types of polymorphism: static and dynamic. Static polymorphism is achieved through method overloading, which allows multiple methods with the same name but different parameters to coexist within a class. Dynamic polymorphism, on the other hand, is achieved through method overriding, which allows a subclass to provide its own implementation of a method defined in its superclass.

Polymorphism is important for several reasons:

Reusability: Polymorphism enables developers to reuse code across different classes and objects, reducing the amount of code that needs to be written and making it easier to maintain and modify code.

Flexibility: Polymorphism allows for greater flexibility in programming, as it allows objects to be treated in a more abstract and general way, rather than being tied to specific types or implementations.

Encapsulation: Polymorphism helps to enforce the principle of encapsulation, which is a fundamental concept in object-oriented programming that promotes the idea of hiding implementation details from the user of a class.

Extensibility: Polymorphism allows for easy extension of existing code, as new classes can be created that implement existing interfaces or inherit from existing classes, without having to modify the original code.

An example of polymorphism is an employee base class, which includes all the basic details about employees. Classes such as clerk and manager could inherit from the employee base class with specific implementations (by overriding virtual methods) wherever necessary in the derived classes.

In summary, polymorphism is an important concept in object-oriented programming that allows for greater flexibility, reusability, encapsulation, and extensibility in software development.

public abstract class Goal // Base class (parent)

{

public virtual void Goal()

{

Console.WriteLine("This is my Goal");

}

}

public class SimpleGoal : Goal // Derived class (child)

{

public override void Goal()

{

Console.WriteLine("This is my Simple Goal");

}

}

public class EternalGoal : Goal // Derived class (child)

{

public override void Goal()

{

Console.WriteLine("This is my Eternal Goal");

}

}

public class ChecklistGoal : Goal // Derived class (child)

{

public override void Goal()

{

Console.WriteLine("This is my Checklist Goal");

}

}

class Program

{

static void Main(string[] args)

{

Goal myGoal = new Goal(); // Create a Goal object

SimpleGoal mySimpleGoal = new SimpleGoal(); // Create a SimpleGoal object

EternalGoal myEternalGoal = new EternalGoal(); // Create a EternalGoal object

ChecklistGoal myCheckListGoal = new ChecklistGoal(); // Create a CheckListGoal object

myGoal.Goal();

mySimpleGoal.Goal();

myEternalGoal.Goal();

myCheckListGoal.Goal();

}

}