**Definition:**

Polymorphism: The third pillar of object-oriented programming, alongside encapsulation and inheritance. It refers to the ability of objects to be treated as instances of their parent class, and the capability of derived classes to provide their own implementation of methods inherited from their parent class.

**Key Aspects:**

**Run-time Polymorphism:** Objects of a derived class can be treated as objects of a base class, allowing for flexibility in method parameters and collections. At runtime, the object's declared type differs from its actual runtime type.

**Method Overriding:** Base classes can define virtual methods, which derived classes can override with their own implementation. This allows for uniform treatment of related objects and enables dynamic method invocation based on the object's actual type.

**Implementation Steps:**

Create a class hierarchy with a base class (Shape) and derived classes (Rectangle, Circle, Triangle).

Define a virtual method (Draw) in the base class and override it in each derived class to draw the specific shape.

Utilize polymorphism by storing derived class objects in a collection of base class type (List<Shape>) and invoking the Draw method on each object, which dynamically calls the appropriate overridden method based on the object's type.

// Define base class Shape

public class Shape

{

// Properties for position and dimensions

public int X { get; private set; }

public int Y { get; private set; }

public int Height { get; set; }

public int Width { get; set; }

// Virtual method for drawing

public virtual void Draw()

{

Console.WriteLine("Performing base class drawing tasks");

}

}

// Define derived classes

public class Circle : Shape

{

public override void Draw()

{

Console.WriteLine("Drawing a circle");

base.Draw();

}

}

public class Rectangle : Shape

{

public override void Draw()

{

Console.WriteLine("Drawing a rectangle");

base.Draw();

}

}

public class Triangle : Shape

{

public override void Draw()

{

Console.WriteLine("Drawing a triangle");

base.Draw();

}

}

// Create instances and demonstrate polymorphism

var shapes = new List<Shape>

{

new Rectangle(),

new Triangle(),

new Circle()

};

foreach (var shape in shapes)

{

shape.Draw();

}

***Notes:***

In C#, every type is polymorphic as all types inherit from Object.

Virtual members enable derived classes to extend or override behavior defined in the base class, providing flexibility and code reuse.