**Mohammad Seyfi Marandi – Exercise 2**

In C#, overriding a method means providing a specific implementation for a method that is already defined in a base class (superclass). By using **override**, a method inherited in a derived class (subclass) can replace the behavior of the original method with its own specialized implementation.

When a method is overridden, it must have the same signature (i.e., the same number and type of parameters) as the original method, and it must also have the same return type. The **override** keyword is used in the method definition in the derived class for this purpose.

Using **override** in C# serves polymorphism, meaning that a method defined in a base class can be called using a reference of the base class type or a reference of a derived class type, and the execution will be directed to the specific implementation in the derived class.

In summary, when a method is overridden in C#, it means that a method with its specific implementation is defined in the derived class to be used instead of the method from the base class for performing a specific operation.

using System;

// Base class

class Shape

{

// Virtual method for drawing the shape

public virtual void Draw()

{

Console.WriteLine("Drawing a generic shape.");

}

}

// Derived class

class Circle : Shape

{

// Override the Draw method to draw a circle

public override void Draw()

{

Console.WriteLine("Drawing a circle.");

}

}

// Derived class

class Square : Shape

{

// Override the Draw method to draw a square

public override void Draw()

{

Console.WriteLine("Drawing a square.");

}

}

class Program

{

static void Main(string[] args)

{

// Create instances of the derived classes

Shape circle = new Circle();

Shape square = new Square();

// Call the Draw method for each object

circle.Draw(); // Output: Drawing a circle.

square.Draw(); // Output: Drawing a square.

}

}

Explanation:

* We have a base class **Shape** with a virtual method **Draw** that provides a generic implementation for drawing shapes.
* We then create two derived classes **Circle** and **Square**, each overriding the **Draw** method with its specific implementation for drawing a circle or a square.
* In the **Main** method, we create instances of the derived classes (**circle** and **square**) using the base class reference.
* When we call the **Draw** method on each object, the overridden version of the method in the appropriate derived class is executed, demonstrating polymorphic behavior.