

Polymorphism Third Pillar of OOP



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Method Overriding



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Agenda

- Method overriding
- Virtual / override keywords



Method Overriding

- Modifying the implementation of an inherited method.



```
public class Shape
{
    public void Draw()
    {
    }
}

public class Circle : Shape
{
}

public class Image : Shape
{
}
```



```
public class Shape
{
    public virtual void Draw()
    {
        // Default implementation
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // New implementation
    }
}
```

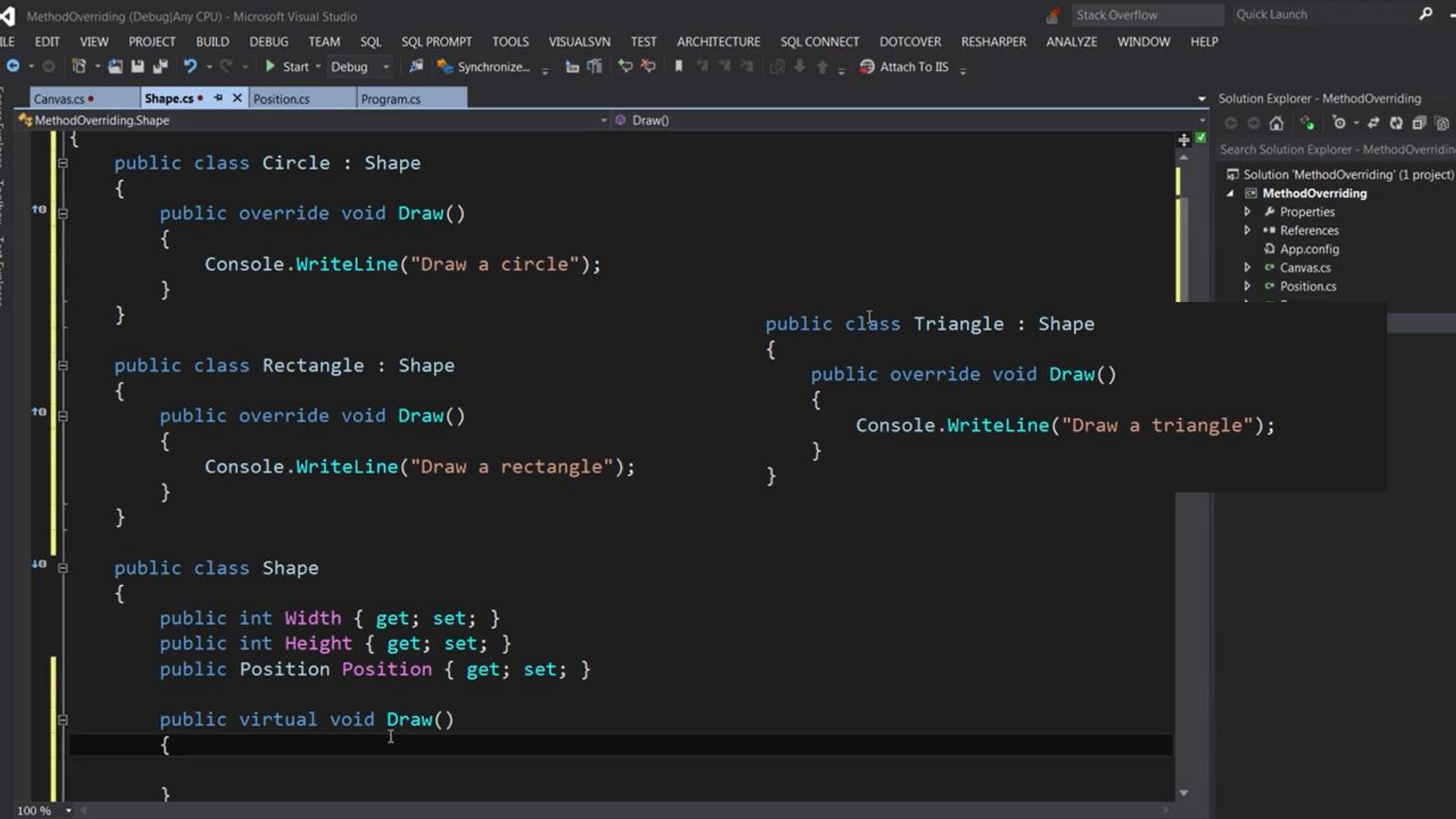


Demo Method Overriding



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular




```
using System;
using System.Collections.Generic;

namespace MethodOverriding
{
    public class Canvas
    {
        public void DrawShapes(List<Shape> shapes)
        {
            foreach (var shape in shapes)
            {
                shape.Draw();
            }
        }
    }
}
```

Search Solution Explorer - MethodOverriding

Solution 'MethodOverriding' (1 project)

MethodOverriding

Properties

References

App.config

Canvas.cs

Position.cs

Program.cs

Shape.cs

```
static void Main(string[] args)
{
    var shapes = new List<Shape>();
    shapes.Add(new Circle());
    shapes.Add(new Rectangle());

    var canvas = new Canvas();
    canvas.DrawShapes(shapes);
}
```

**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Abstract Classes and Members



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Agenda

- Abstract modifier
- Rules about abstract classes and members



Abstract Modifier

- Indicates that a class or a member is missing implementation.



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

```
public class Shape
{
    public virtual void Draw()
    {
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}
```



```
public abstract class Shape
{
    public abstract void Draw();
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```



Abstract Members

- Do not include implementation.

```
public abstract void Draw();
```



Abstract Members

- If a member is declared as abstract, the containing class needs to be declared as abstract too.

```
public abstract class Shape
{
    public abstract void Draw();
}
```



Derived Classes

- Must implement all abstract members in the base abstract class.

```
public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```



Abstract Classes

- Cannot be instantiated.

```
var shape = new Shape(); // Won't compile
```



Why to use Abstract?

- When you want to provide some common behaviour, while forcing other developers to follow your design.



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

```
public abstract class Shape
{
    public abstract void Draw();
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```



Demo Abstract Classes and Members



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

```
public class Shape
{
    public int Width { get; set; }
    public int Height { get; set; }

    public virtual void Draw()
    {

    }
}
```

Toolbox Test Explorer

AbstractClasses

- Properties
- References
- App.config
- Program.cs
- Shape.cs

```
namespace AbstractClasses
{
    public class Circle : Shape
    {
        public override void Draw()
        {
            Console.WriteLine("Draw a circle");
        }
    }
}
```

Toolbox Test Explorer

AbstractClasses

- Properties
- References
- App.config
- Circle.cs
- Program.cs
- Shape.cs



AbstractClasses (Debug)Any CPU - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL SQL PROMPT TOOLS VISUALSVN TEST ARCHITECTURE SQL CONNECT DOTCOVER RESHARPER ANALYZE WINDOW HELP

Start Debug Synchronize Attach To IIS

Circle.cs Program.cs Shape.cs

AbstractClasses.Program Main(string[] args)

```
namespace AbstractClasses
{
    public class Rectangle : Shape
    {
    }

    class Program
    {
        static void Main(string[] args)
        {
            var circle = new Circle();
            circle.Draw();

            var rectangle = new Rectangle();
            rectangle.Draw();
        }
    }
}
```

Solution Explorer - AbstractClasses

Search Solution Explorer - AbstractClasses

Solution 'AbstractClasses' (1 project)

- AbstractClasses
 - Properties
 - References
 - App.config
 - Circle.cs
 - Program.cs
 - Shape.cs

C:\Windows\system32\cmd.exe

Draw a circle
Press any key to continue . . .



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

C# Intermediate

AbstractClasses (Debug)Any CPU - Microsoft Visual Studio

Stack Overflow Quick Launch

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL SQL PROMPT TOOLS VISUALSVN TEST ARCHITECTURE SQL CONNECT DOTCOVER RESHARPER ANALYZE WINDOW HELP

Start Debug Synchronize Attach To IIS

Circle.cs Program.cs Shape.cs

AbstractClasses.Shape Select()

```
using System;

namespace AbstractClasses
{
    public abstract class Shape
    {
        public int Width { get; set; }
        public int Height { get; set; }

        public abstract void Draw();

        public void Copy()
        {
            Console.WriteLine("Copy shape into clipboard.");
        }

        public void Select()
        {
            Console.WriteLine("Select the shape.");
        }
    }
}
```

Solution Explorer - AbstractClasses

Search Solution Explorer - AbstractClasses

Solution 'AbstractClasses' (1 project)

- AbstractClasses
 - Properties
 - References
 - App.config
 - Circle.cs
 - Program.cs
 - Shape.cs




```
using System;

namespace AbstractClasses
{
    public class Rectangle : Shape
    {
        public override void Draw()
        {
            Console.WriteLine("Draw a rectangle");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var circle = new Circle();
            circle.Draw();

            var rectangle = new Rectangle();
            rectangle.Draw();
        }
    }
}
```

- Properties
- References
- App.config
- Circle.cs
- Program.cs
- Shape.cs

**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Sealed Classes and Members



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Sealed Modifier

- Prevents derivation of classes or overriding of methods.



```
public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}
```



```
public sealed class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}
```



Why?

- Sealed classes are slightly faster because of some run-time optimizations.



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Exercises



Zouhair Rimale, Ph.D.

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Exercise 1 : Design a database connection

To access a database, we need to open a connection to it first and close it once our job is done.

Connecting to a database depends on the type of the target database and the database management system (DBMS). For example, connecting to a SQL Server database is different from connecting to an Oracle database. But both these connections have a few things in common:

- They have a connection string
- They can be opened
- They can be closed
- They may have a timeout attribute (so if the connection could not be opened within the timeout, an exception will be thrown).

Your job is to represent these commonalities in a base class called **DbConnection**. This class should have two properties:

ConnectionString : string

Timeout : TimeSpan

A **DbConnection** will not be in a valid state if it doesn't have a connection string. So you need to pass a connection string in the constructor of this class. Also, take into account the scenarios where null or an empty string is sent as the connection string. Make sure to throw an exception to guarantee that your class will always be in a valid state.

Our **DbConnection** should also have two methods for opening and closing a connection. We don't know how to open or close a connection in a **DbConnection** and this should be left to the classes that derive from **DbConnection**. These classes (eg **SqlConnection** or **OracleConnection**) will provide the actual implementation. So you need to declare these methods as abstract.

Derive two classes **SqlConnection** and **OracleConnection** from **DbConnection** and provide a simple implementation of opening and closing connections using `Console.WriteLine()`. In the real-world, SQL Server provides an API for opening or closing a connection to a database. But for this exercise, we don't need to worry about it.

Exercise 2 : Design a database command

Now that we have the concept of a **DbConnection**, let's work out how to represent a DbCommand.

Design a class called **DbCommand** for executing an instruction against the database. A **DbCommand** cannot be in a valid state without having a connection. So in the constructor of this class, pass a **DbConnection**. Don't forget to cater for the null.

Each **DbCommand** should also have the instruction to be sent to the database. In case of SQL Server, this instruction is expressed in T-SQL language. Use a string to represent this instruction. Again, a command cannot be in a valid state without this instruction. So make sure to receive it in the constructor and cater for the null reference or an empty string.

Each command should be executable. So we need to create a method called **Execute()**. In this method, we need a simple implementation as follows:

- Open the connection

- Run the instruction

- Close the connection

Note that here, inside the **DbCommand**, we have a reference to **DbConnection**. Depending on the type of **DbConnection** sent at runtime, opening and closing a connection will be different. For example, if we initialize this **DbCommand** with a **SqlConnection**, we will open and close a connection to a Sql Server database. This is polymorphism. Interestingly, **DbCommand** doesn't care about how a connection is opened or closed. It's not the responsibility of the **DbCommand**.

All it cares about is to send an instruction to a database.

For running the instruction, simply output it to the Console. In the real-world, SQL Server (or any other DBMS) provides an API for running an instruction against the database. We don't need to worry about it for this exercise.

In the main method, initialize a **DbCommand** with some string as the instruction and a **SqlConnection**. Execute the command and see the result on the console.

Then, swap the **SqlConnection** with an **OracleConnection** and see polymorphism in action.