目　录

一、实验内容

对于作业 2 的计算器，采用 IOC 的设计思想进行重构

使用 Autofac 框架完成日志记录器对象和窗体对象的实例化，使用三种方式实现

（1）设计静态功能类，在静态功能类中完成组件注册

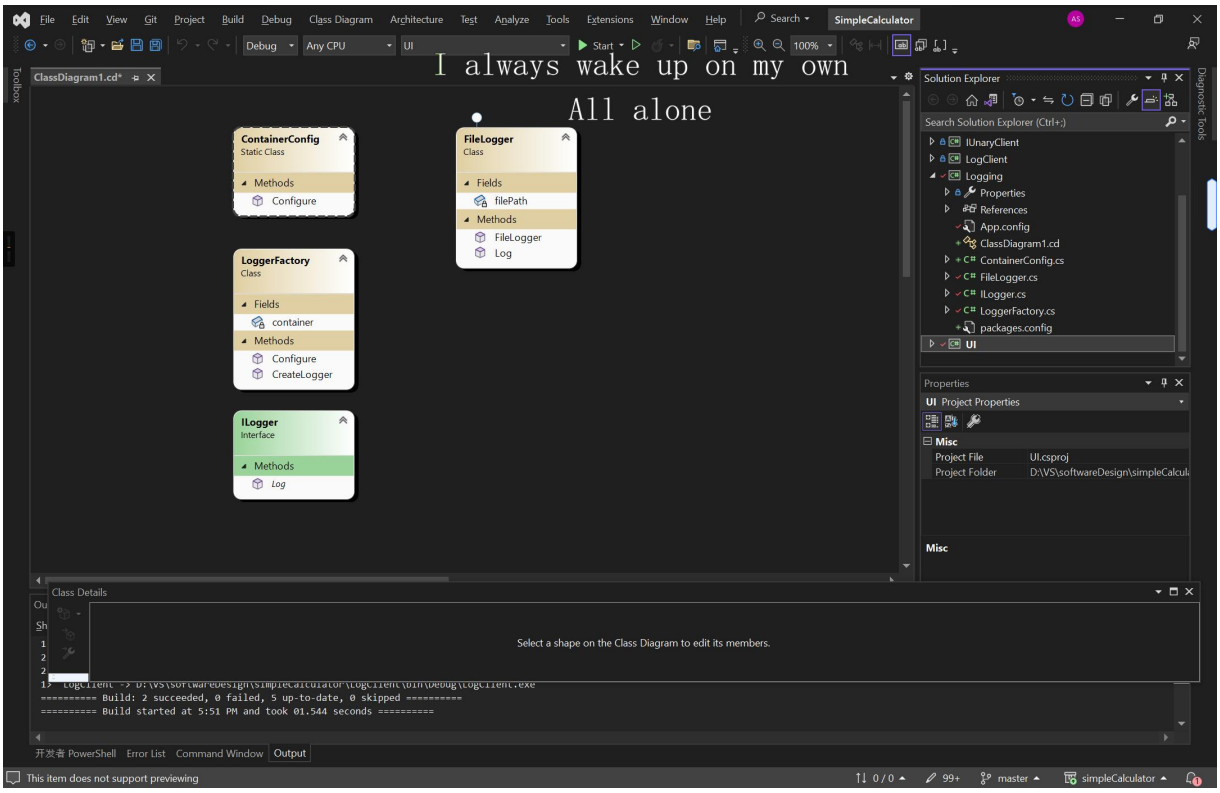
（2）使用配置文件 app.config （自定义配置节 autofac、在 autofac 中配置要注册的组件，通过 RegistModule 注册组件）

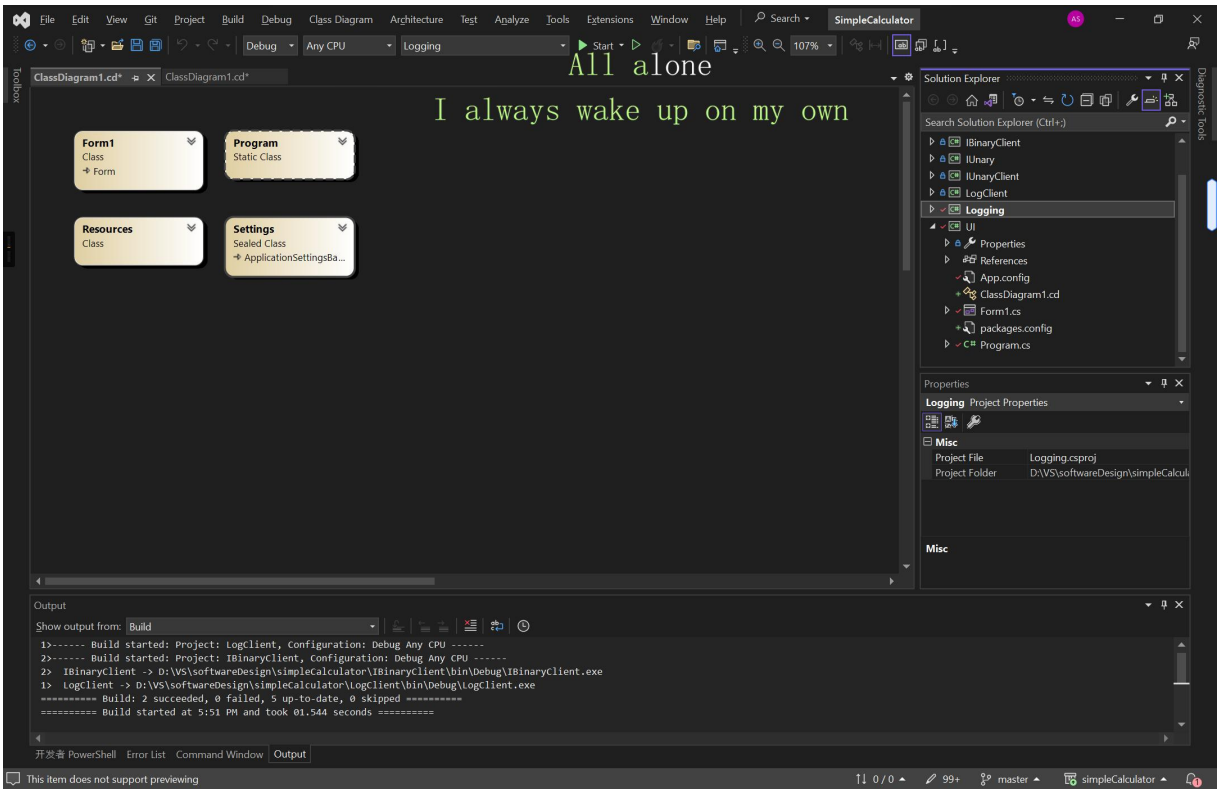
（3）使用配置文件 xx.json or xx.XML 配置要要注册的组件

二、类图及代码结构

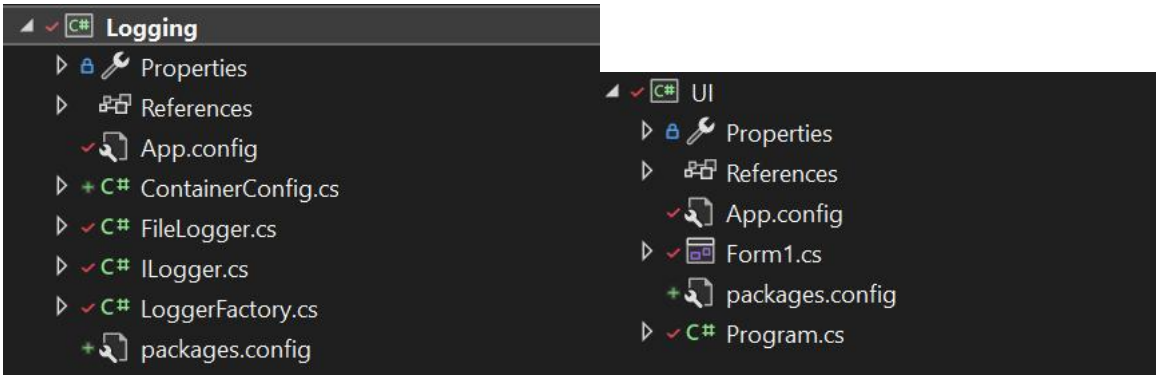
（1）设计静态功能类，在静态功能类中完成组件注册

1. 类图



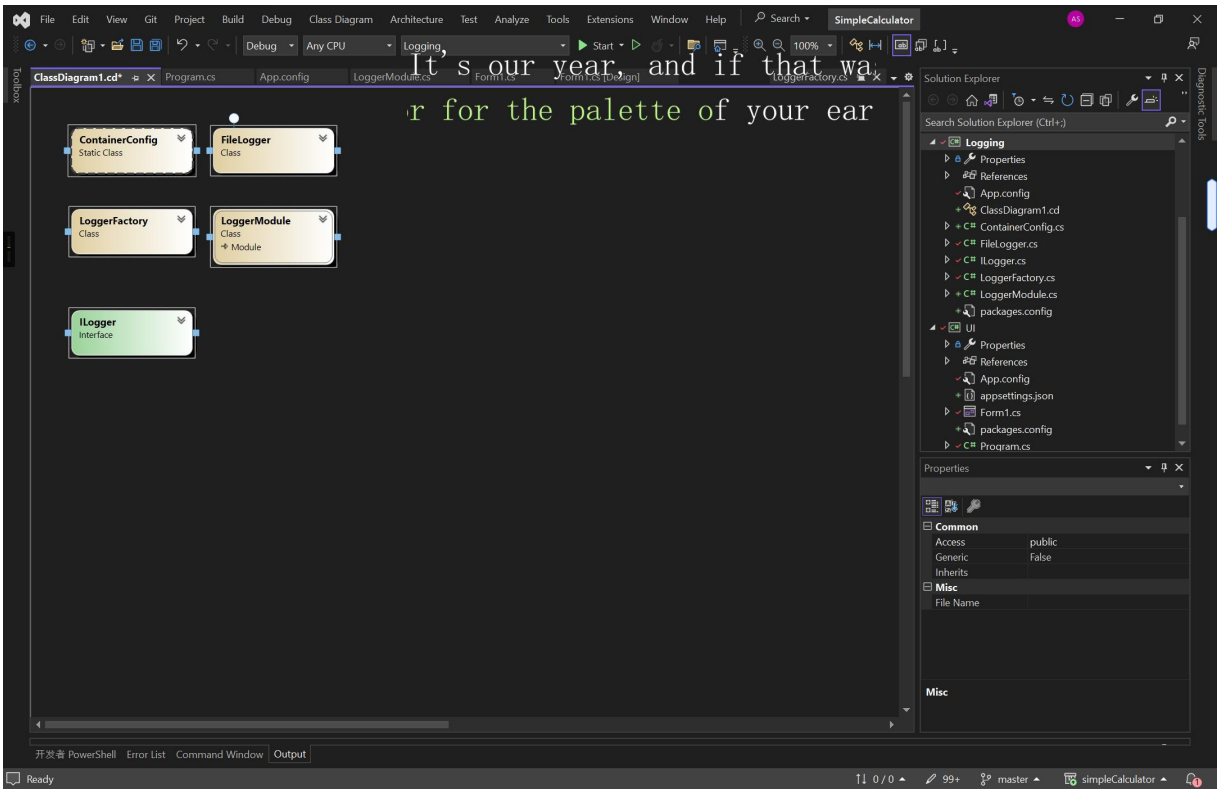


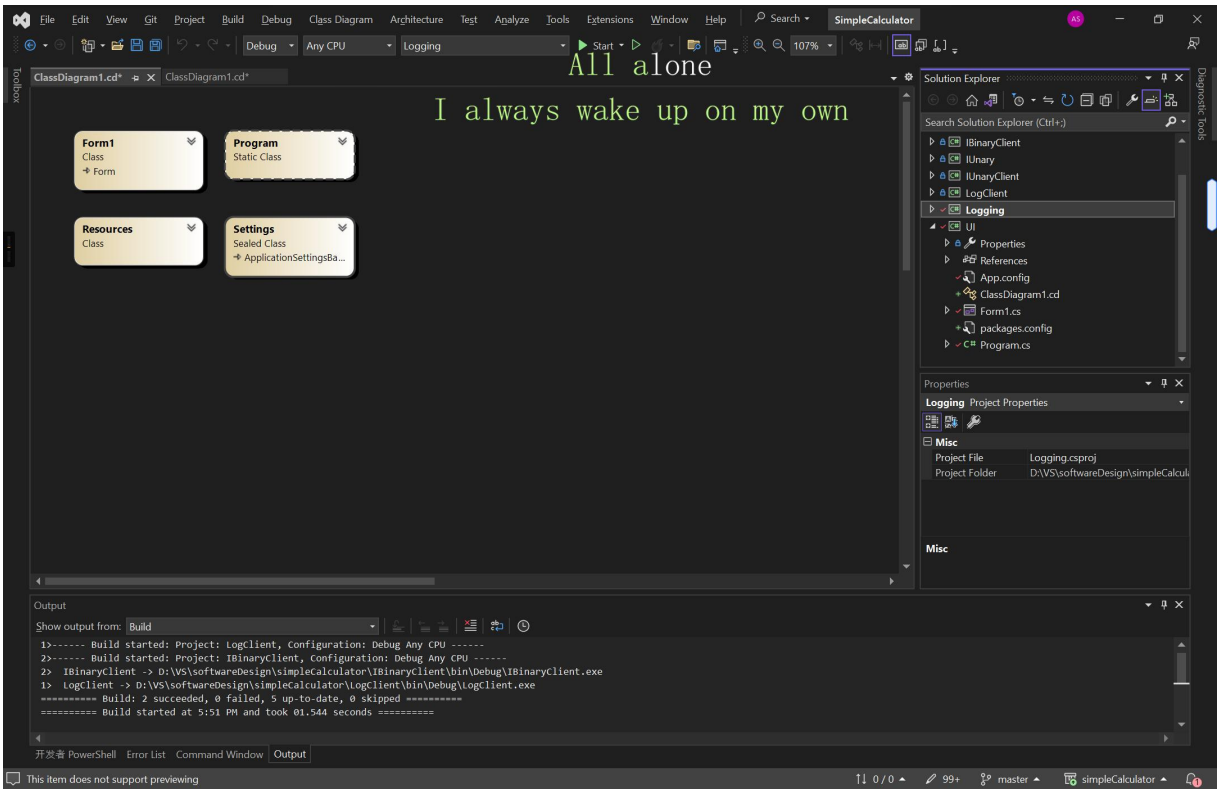
2. 代码结构



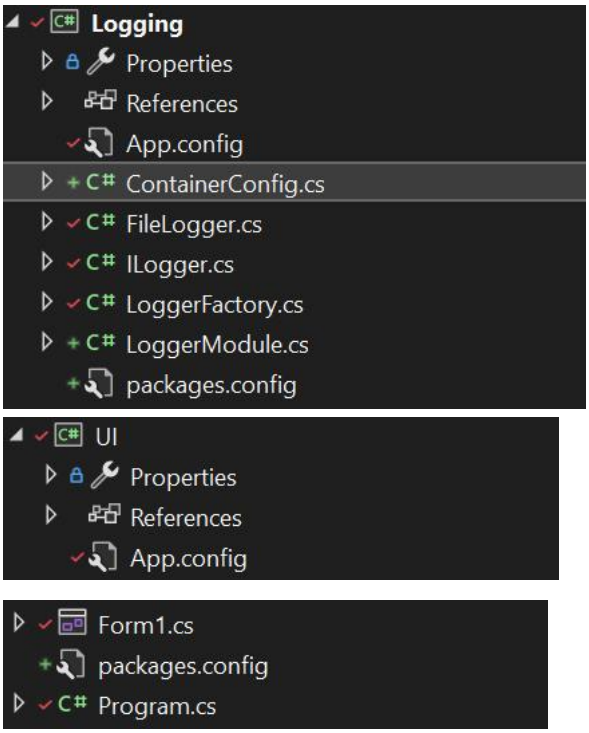
（2）使用配置文件 app.config （自定义配置节 autofac、在 autofac 中配置要注册的组件，通过 RegistModule 注册组件）

1. 类图


It's our year, and if that wa
r for the palette of your ear
ContainerConfig
Static Class
FileLogger
Class
LoggerFactory
Class
LoggerModule
Class
Module
ILogger
Interface

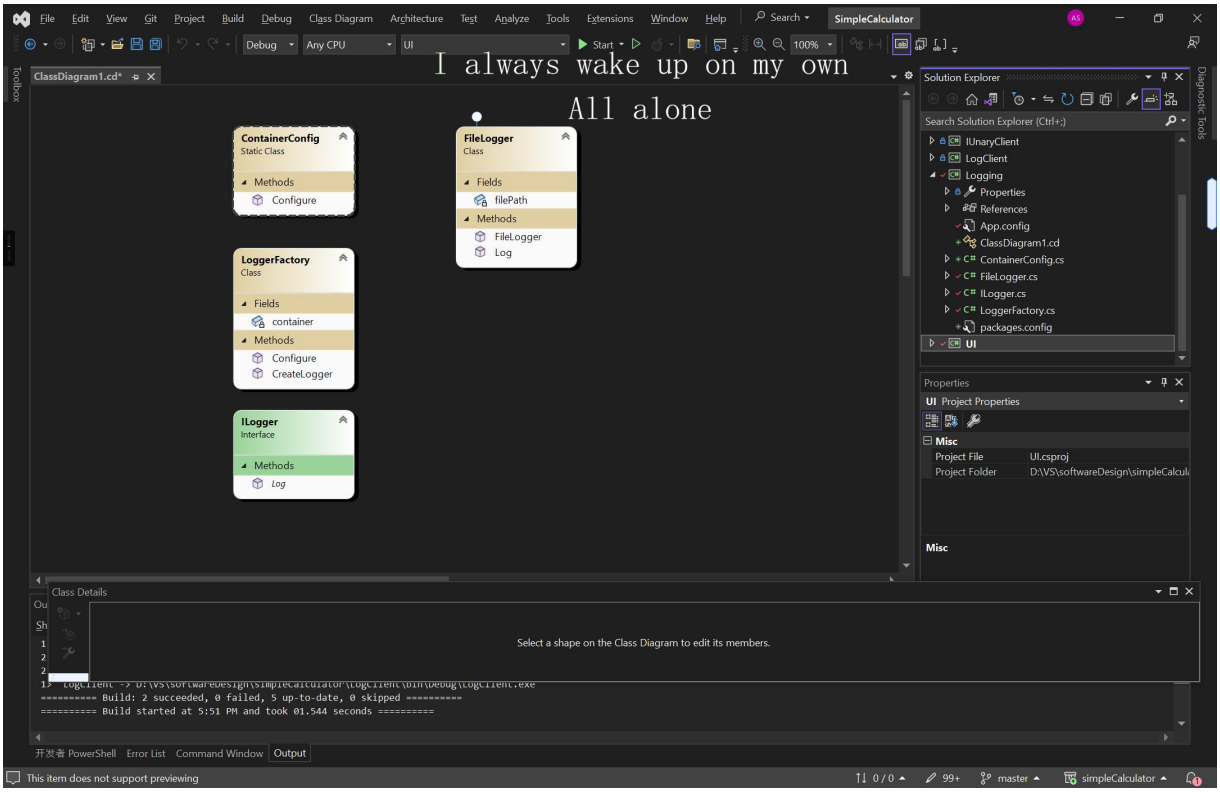

All alone
I always wake up on my own
Form1
Class
Form
Program
Static Class
Resources
Class
Settings
Sealed Class
1>------ Build started: Project: LogClient, Configuration: Debug Any CPU ------
2>------ Build started: Project: IBinaryClient, Configuration: Debug Any CPU ------
2> IBinaryClient -> D:\VS\softwareDesign\simpleCalculator\IBinaryClient\bin\Debug\IBinaryClient.exe
1> LogClient -> D:\VS\softwareDesign\simpleCalculator\LogClient\bin\Debug\LogClient.exe
========== Build: 2 succeeded, 0 failed, 5 up-to-date, 0 skipped ==========
========== Build started at 5:51 PM and took 01.544 seconds ==========

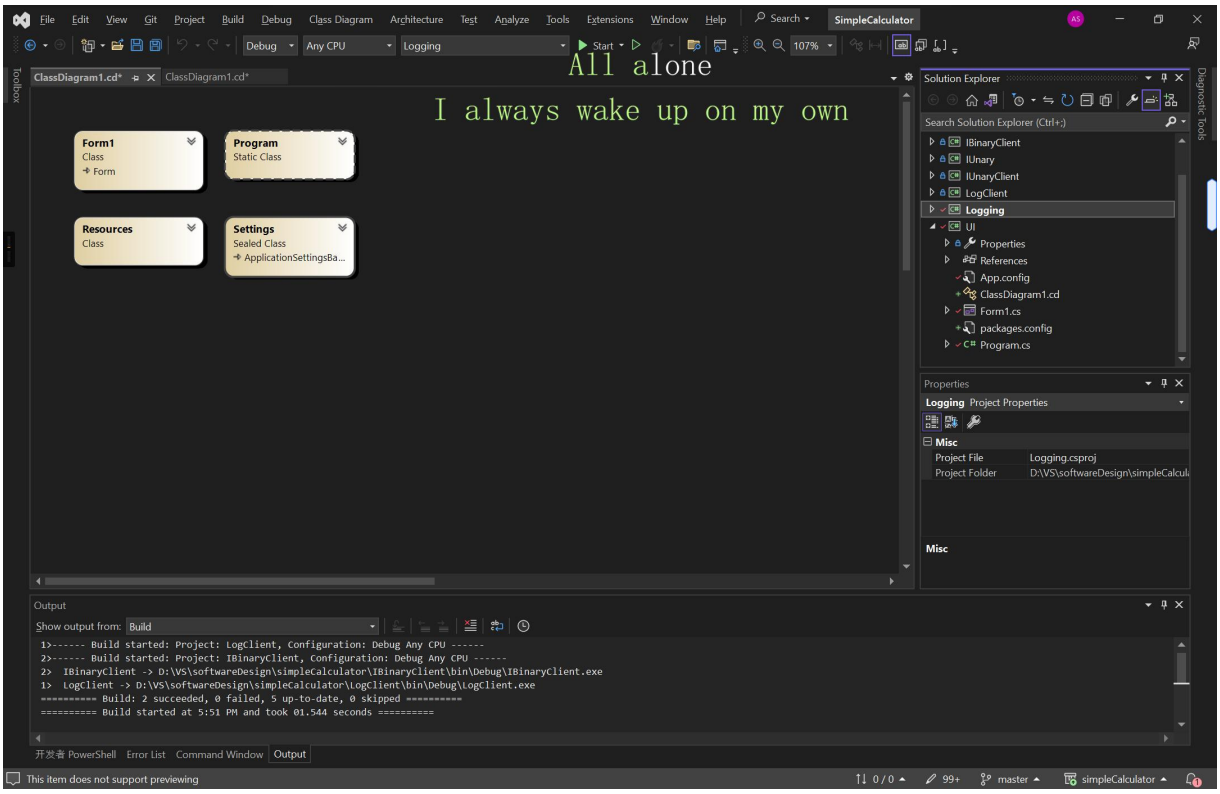
2. 代码结构



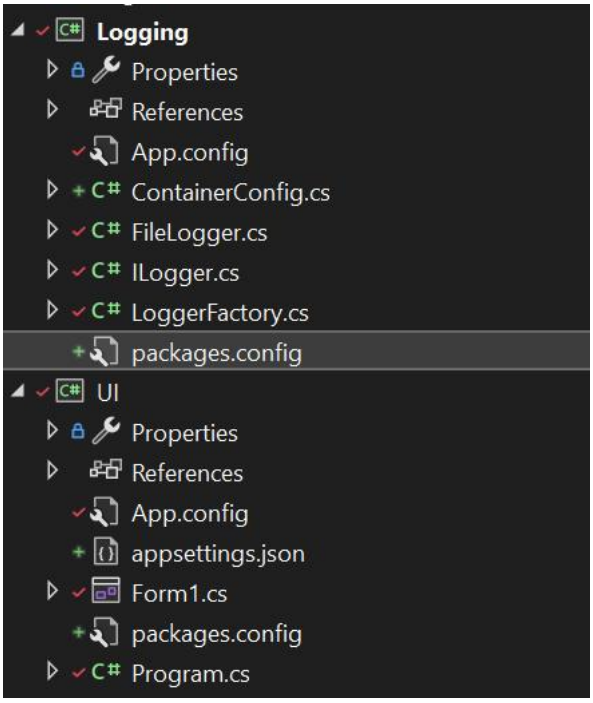
（3）使用配置文件 xx.json or xx.XML 配置要要注册的组件

1. 类图



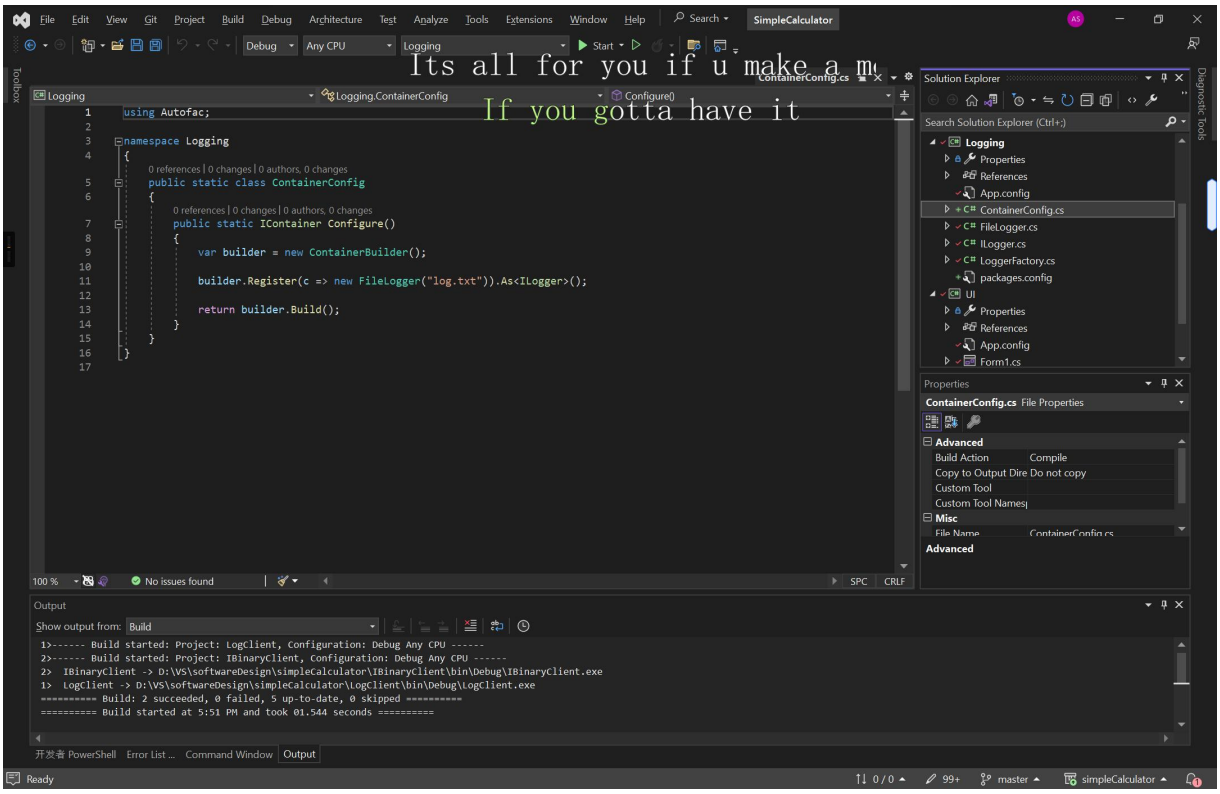


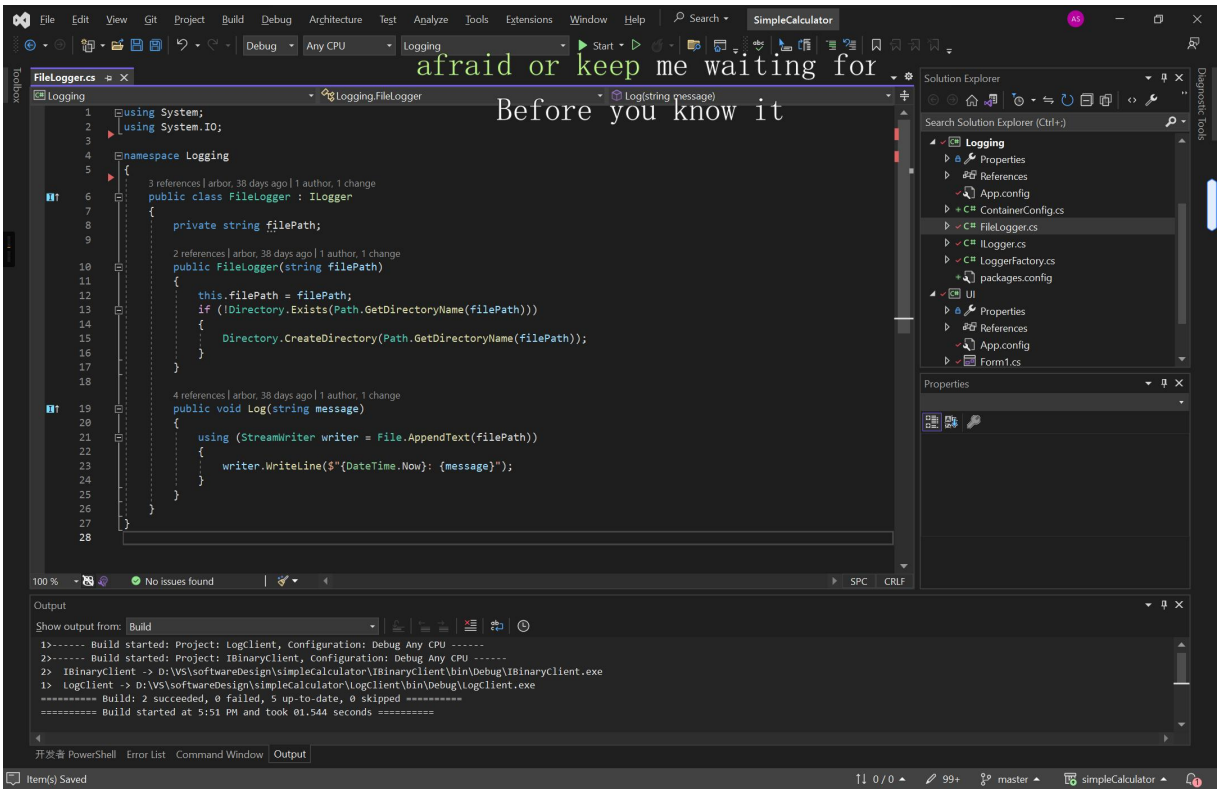
2. 代码结构

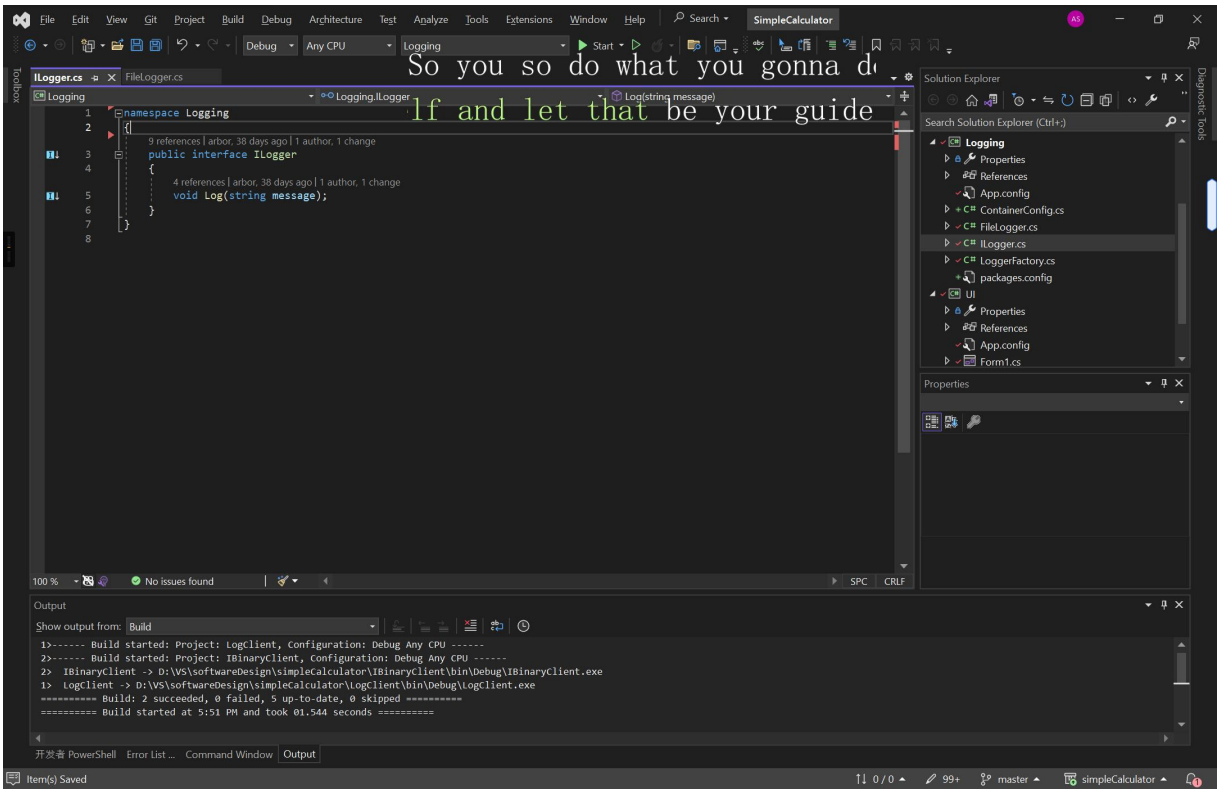


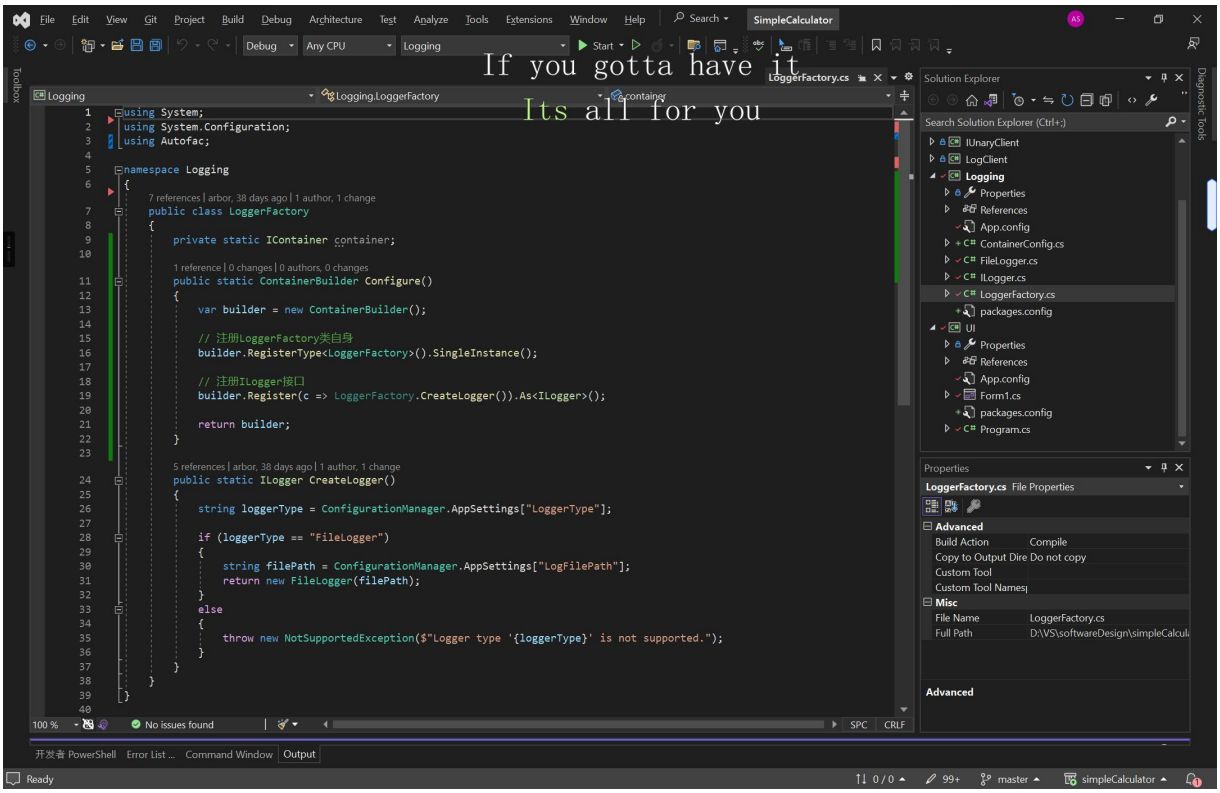
三、代码实现

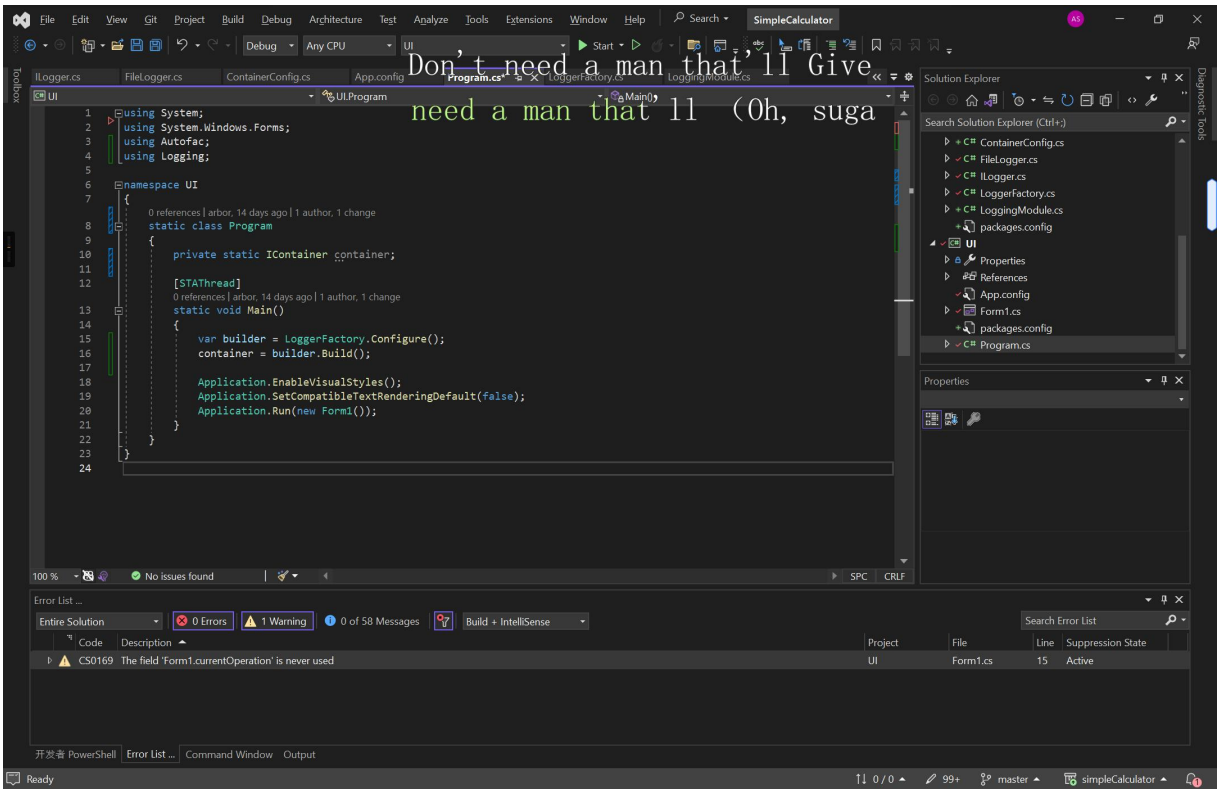
（1）设计静态功能类，在静态功能类中完成组件注册

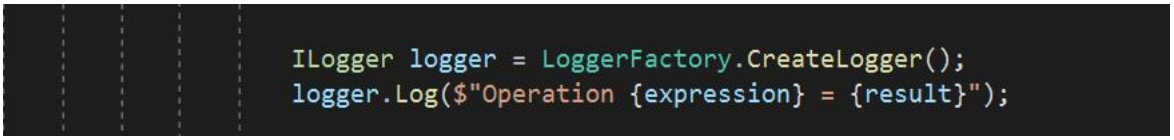

Its all for you if u make a m
If you gotta have it
ContainerConfig.cs
Logging
Logging.ContainerConfig
Configure()
using Autofac;
namespace Logging
{
0 references | 0 changes | 0 authors, 0 changes
public static class ContainerConfig
{
0 references | 0 changes | 0 authors, 0 changes
public static IContainer Configure()
{
var builder = new ContainerBuilder();
builder.Register(c => new FileLogger("log.txt")).As<ILogger>();
return builder.Build();
}
}
}
Solution Explorer
Logging
Properties
References
App.config
ContainerConfig.cs
FileLogger.cs
ILogger.cs
LoggerFactory.cs
packages.config
UI
Properties
References
App.config
Form1.cs
Properties
ContainerConfig.cs File Properties
Advanced
Build Action
Compile
Copy to Output Dire
Do not copy
Custom Tool
Custom Tool Names
Misc
Advanced
No issues found
Output
Show output from:
Build
1>------ Build started: Project: LogClient, Configuration: Debug Any CPU ------
2>------ Build started: Project: IBinaryClient, Configuration: Debug Any CPU ------
2> IBinaryClient -> D:\VS\softwareDesign\simpleCalculator\IBinaryClient\bin\Debug\IBinaryClient.exe
1> LogClient -> D:\VS\softwareDesign\simpleCalculator\LogClient\bin\Debug\LogClient.exe
========== Build: 2 succeeded, 0 failed, 5 up-to-date, 0 skipped ==========
========== Build started at 5:51 PM and took 01.544 seconds ==========
Ready
master
simpleCalculator


afraid or keep me waiting for
Before you know it
FileLogger.cs
Logging
Logging.FileLogger
Log(string message)
using System;
using System.IO;
namespace Logging
{
3 references | arbor, 38 days ago | 1 author, 1 change
public class FileLogger : ILogger
{
private string filePath;
2 references | arbor, 38 days ago | 1 author, 1 change
public FileLogger(string filePath)
{
this.filePath = filePath;
if (!Directory.Exists(Path.GetDirectoryName(filePath)))
{
Directory.CreateDirectory(Path.GetDirectoryName(filePath));
}
}
4 references | arbor, 38 days ago | 1 author, 1 change
public void Log(string message)
{
using (StreamWriter writer = File.AppendText(filePath))
{
writer.WriteLine($"{DateTime.Now}: {message}");
}
}
}
}
Solution Explorer
Logging
Properties
References
App.config
ContainerConfig.cs
FileLogger.cs
ILogger.cs
LoggerFactory.cs
packages.config
UI
Properties
References
App.config
Form1.cs
Properties
No issues found
Output
Show output from:
Build
1>------ Build started: Project: LogClient, Configuration: Debug Any CPU ------
2>------ Build started: Project: IBinaryClient, Configuration: Debug Any CPU ------
2> IBinaryClient -> D:\VS\softwareDesign\simpleCalculator\IBinaryClient\bin\Debug\IBinaryClient.exe
1> LogClient -> D:\VS\softwareDesign\simpleCalculator\LogClient\bin\Debug\LogClient.exe
========== Build: 2 succeeded, 0 failed, 5 up-to-date, 0 skipped ==========
========== Build started at 5:51 PM and took 01.544 seconds ==========
Item(s) Saved
master
simpleCalculator


So you so do what you gonna d
If and let that be your guide
namespace Logging
{
9 references | arbor, 38 days ago | 1 author, 1 change
public interface ILogger
{
4 references | arbor, 38 days ago | 1 author, 1 change
void Log(string message);
}
}
Solution Explorer
Logging
Properties
References
App.config
ContainerConfig.cs
FileLogger.cs
ILogger.cs
LoggerFactory.cs
packages.config
UI
Properties
References
App.config
Form1.cs
Output
Show output from: Build
1>------ Build started: Project: LogClient, Configuration: Debug Any CPU ------
2>------ Build started: Project: IBinaryClient, Configuration: Debug Any CPU ------
2> IBinaryClient -> D:\VS\softwareDesign\simpleCalculator\IBinaryClient\bin\Debug\IBinaryClient.exe
1> LogClient -> D:\VS\softwareDesign\simpleCalculator\LogClient\bin\Debug\LogClient.exe
========== Build: 2 succeeded, 0 failed, 5 up-to-date, 0 skipped ==========
========== Build started at 5:51 PM and took 01.544 seconds ==========
Item(s) Saved

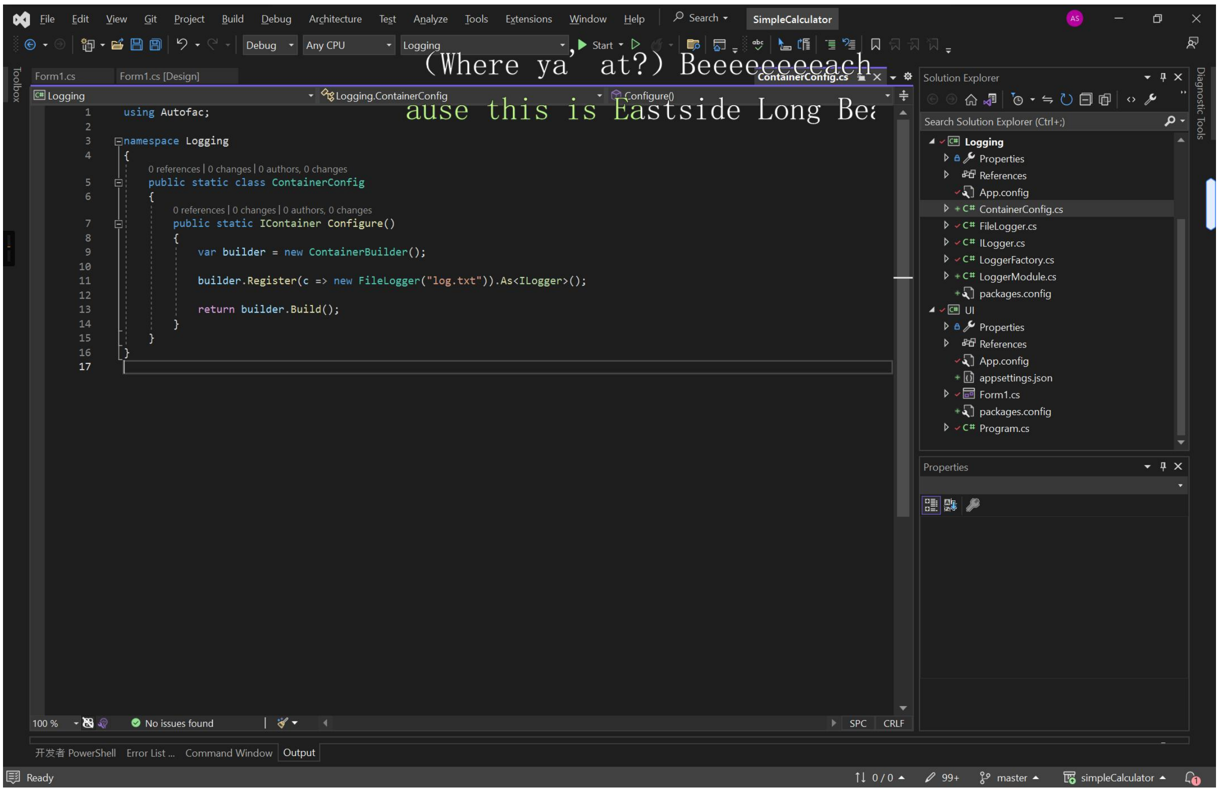

If you gotta have it
Its all for you
using System;
using System.Configuration;
using Autofac;
namespace Logging
{
7 references | arbor, 38 days ago | 1 author, 1 change
public class LoggerFactory
{
private static IContainer container;
1 reference | 0 changes | 0 authors, 0 changes
public static ContainerBuilder Configure()
{
var builder = new ContainerBuilder();
// 注册LoggerFactory类自身
builder.RegisterType<LoggerFactory>().SingleInstance();
// 注册ILogger接口
builder.Register(c => LoggerFactory.CreateLogger()).As<ILogger>();
return builder;
}
5 references | arbor, 38 days ago | 1 author, 1 change
public static ILogger CreateLogger()
{
string loggerType = ConfigurationManager.AppSettings["LoggerType"];
if (loggerType == "FileLogger")
{
string filePath = ConfigurationManager.AppSettings["LogFilePath"];
return new FileLogger(filePath);
}
else
{
throw new NotSupportedException($"Logger type '{loggerType}' is not supported.");
}
}
}
}
Solution Explorer
IUnaryClient
LogClient
Logging
Properties
References
App.config
ContainerConfig.cs
FileLogger.cs
ILogger.cs
LoggerFactory.cs
packages.config
UI
Properties
References
App.config
Form1.cs
packages.config
Program.cs
Properties
LoggerFactory.cs File Properties
Advanced
Build Action Compile
Copy to Output Dire Do not copy
Custom Tool
Custom Tool Namesp
Misc
File Name LoggerFactory.cs
Full Path D:\VS\softwareDesign\simpleCalcul
Advanced
Ready

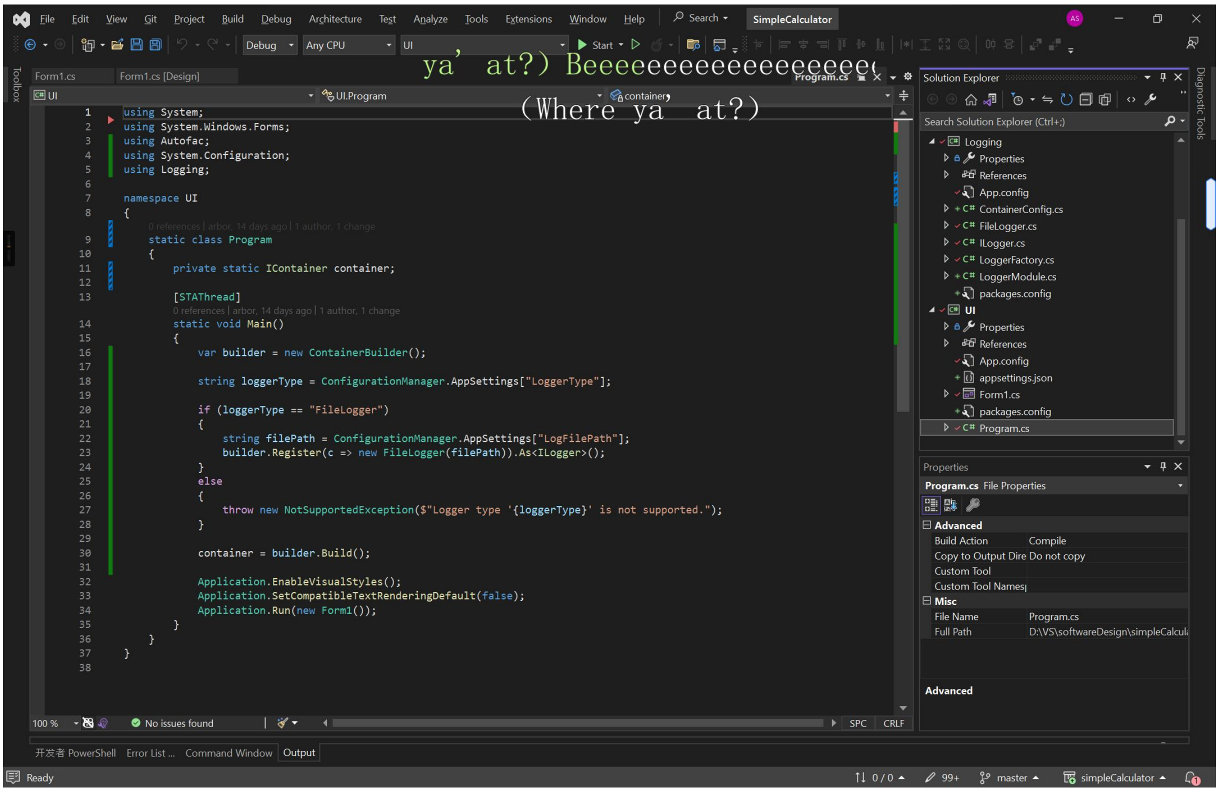

Don’t need a man that’ll Give
need a man that ll (Oh, suga
using System;
using System.Windows.Forms;
using Autofac;
using Logging;
namespace UI
{
static class Program
{
private static IContainer container;
[STAThread]
static void Main()
{
var builder = LoggerFactory.Configure();
container = builder.Build();
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
}
}
}
Solution Explorer
ContainerConfig.cs
FileLogger.cs
ILogger.cs
LoggerFactory.cs
LoggingModule.cs
packages.config
UI
Properties
References
App.config
Form1.cs
packages.config
Program.cs
Error List ...
0 Errors
1 Warning
0 of 58 Messages
Build + IntelliSense
CS0169 The field 'Form1.currentOperation' is never used
UI
Form1.cs
15
Active
Ready


ILogger logger = LoggerFactory.CreateLogger();
logger.Log($"Operation {expression} = {result}");

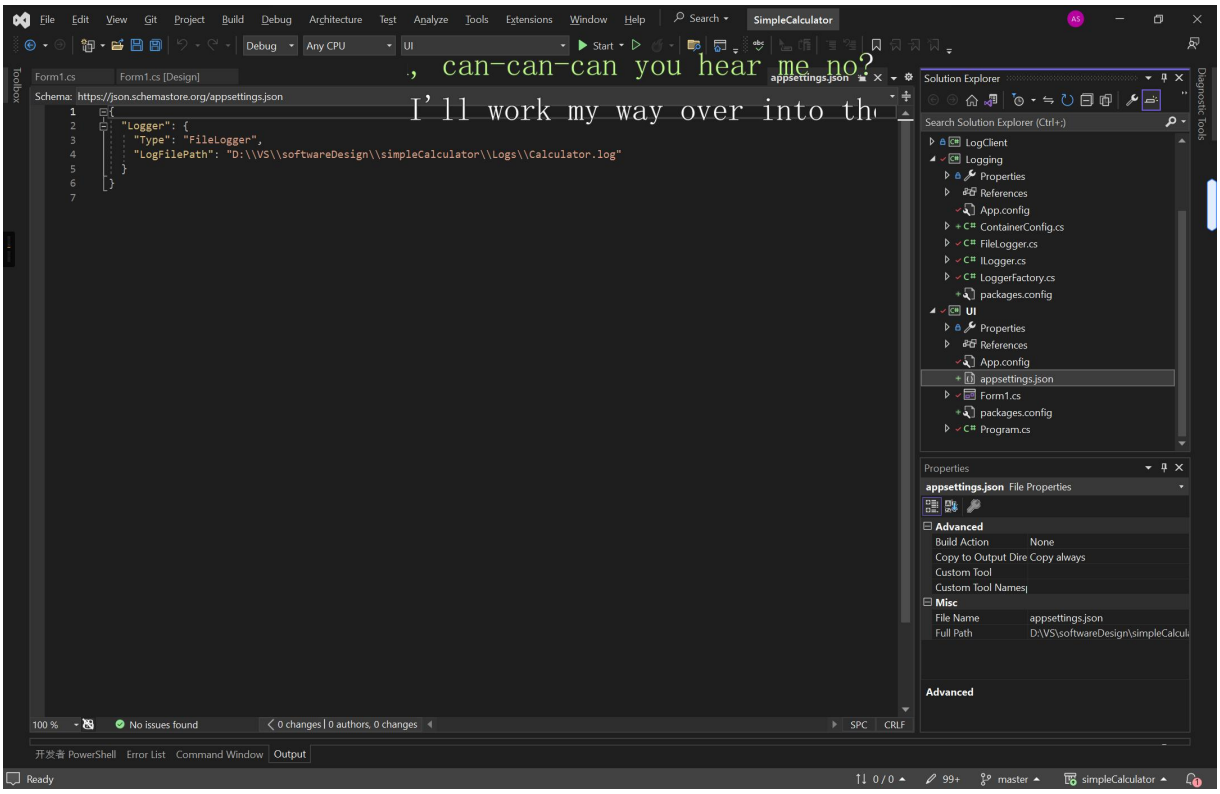
```
    private void logToolStripMenuItem_Click(object sender, EventArgs e)
    {
        ILogger logger = LoggerFactory.CreateLogger();
        string logContent = GetLogContentFromLogger(logger); // 从日志记录器对象获取日志内容

        // 显示日志信息
        MessageBox.Show(logContent, "Log Information", MessageBoxButtons.OK, MessageBoxIcon.Informatio
    }
    1 reference | 0 changes | 0 authors, 0 changes
    private string GetLogContentFromLogger(ILogger logger)
    {
        // 从配置文件中读取日志文件路径
        string logFilePath = ConfigurationManager.AppSettings["LogFilePath"];

        // 检查日志文件是否存在
        if (!File.Exists(logFilePath))
        {
            return "Log file does not exist.";
        }

        try
        {
            // 读取日志文件内容
            string logContent = File.ReadAllText(logFilePath);

            return logContent;
        }
        catch (IOException ex)
        {
            // 处理读取文件异常
            return $"Failed to read log file: {ex.Message}";
        }
    }
}
```

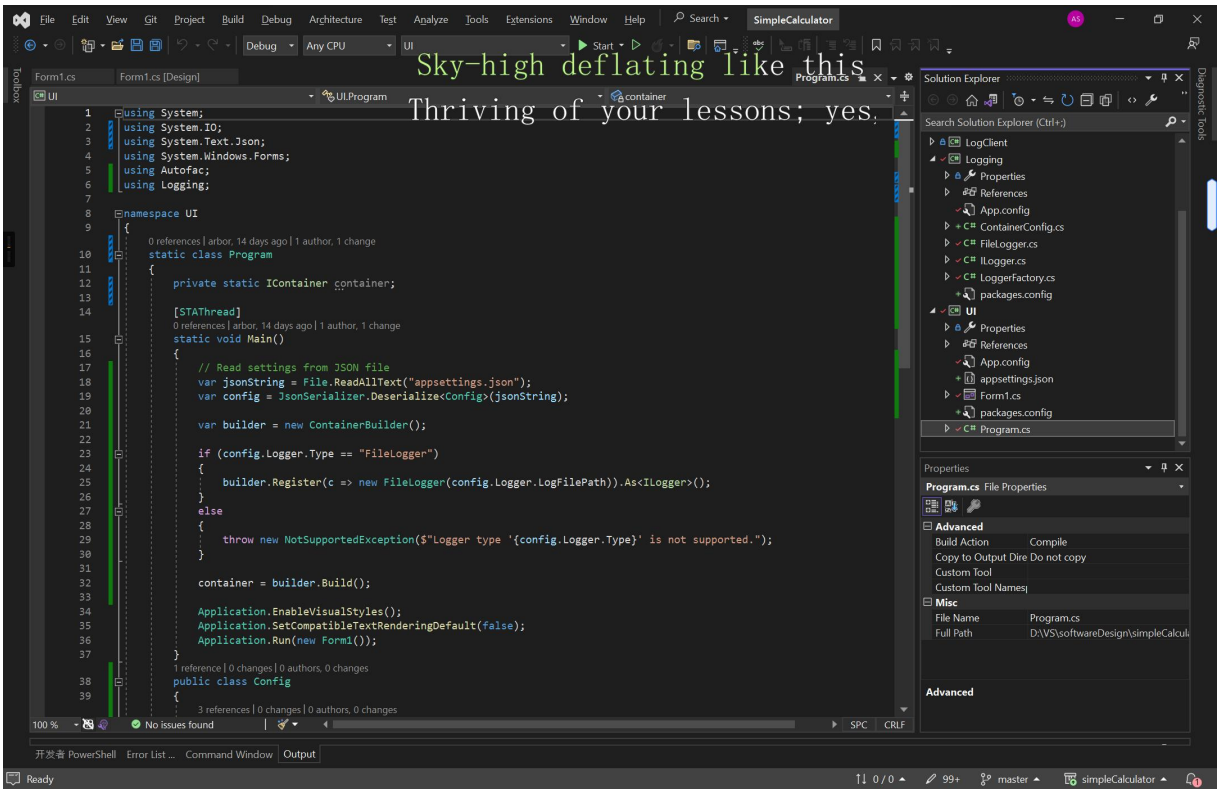
（2）使用配置文件 app.config （自定义配置节 autofac、在 autofac 中配置要注册的组件，通过 RegistModule 注册组件）

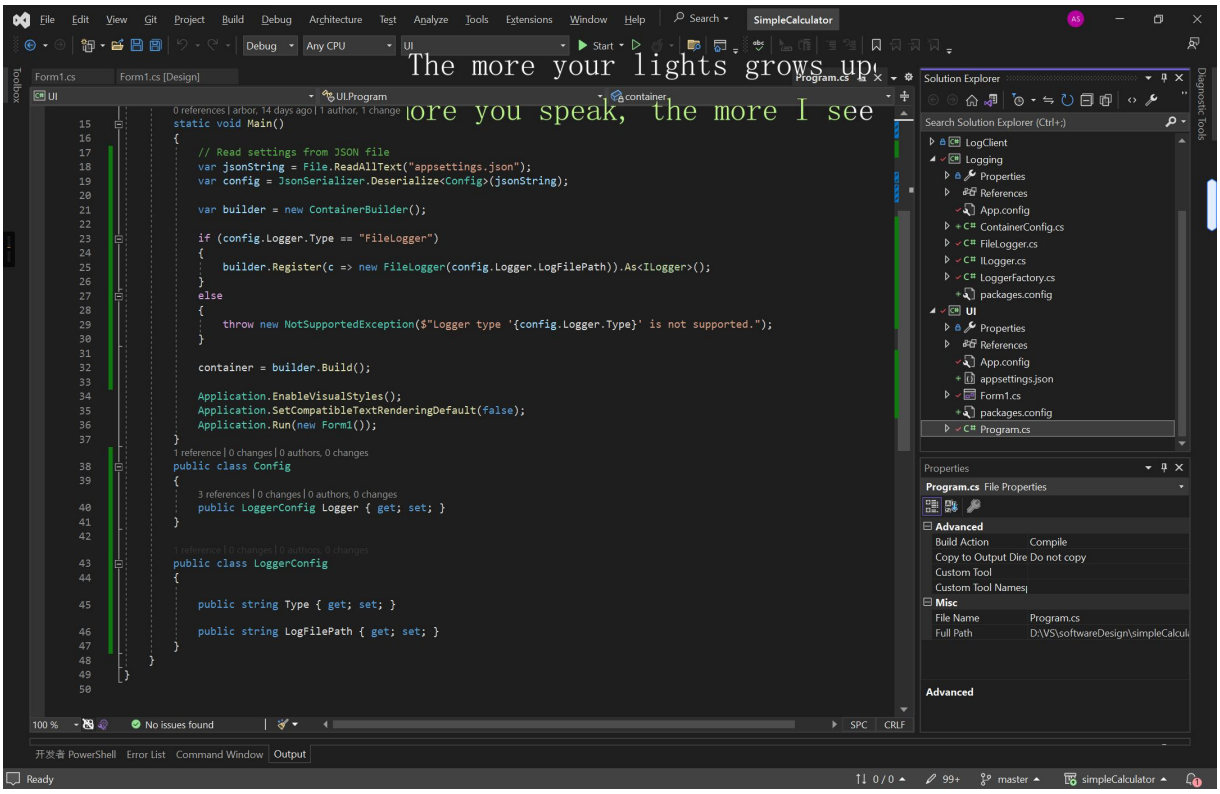




（3）使用配置文件 xx.json or xx.XML 配置要要注册的组件

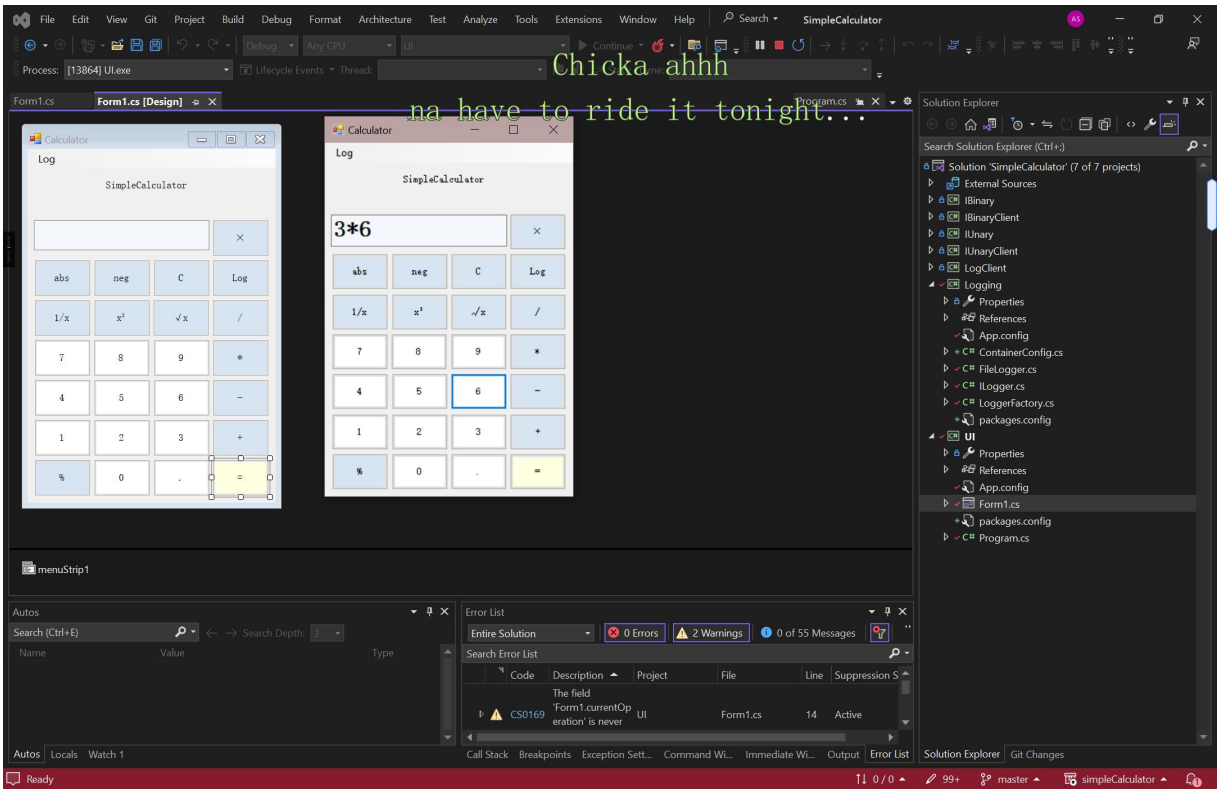

, can-can-can you hear me no?
I'll work my way over into th
"Logger": {
"Type": "FileLogger",
"LogFilePath": "D:\\VS\\softwareDesign\\simpleCalculator\\Logs\\Calculator.log"
}

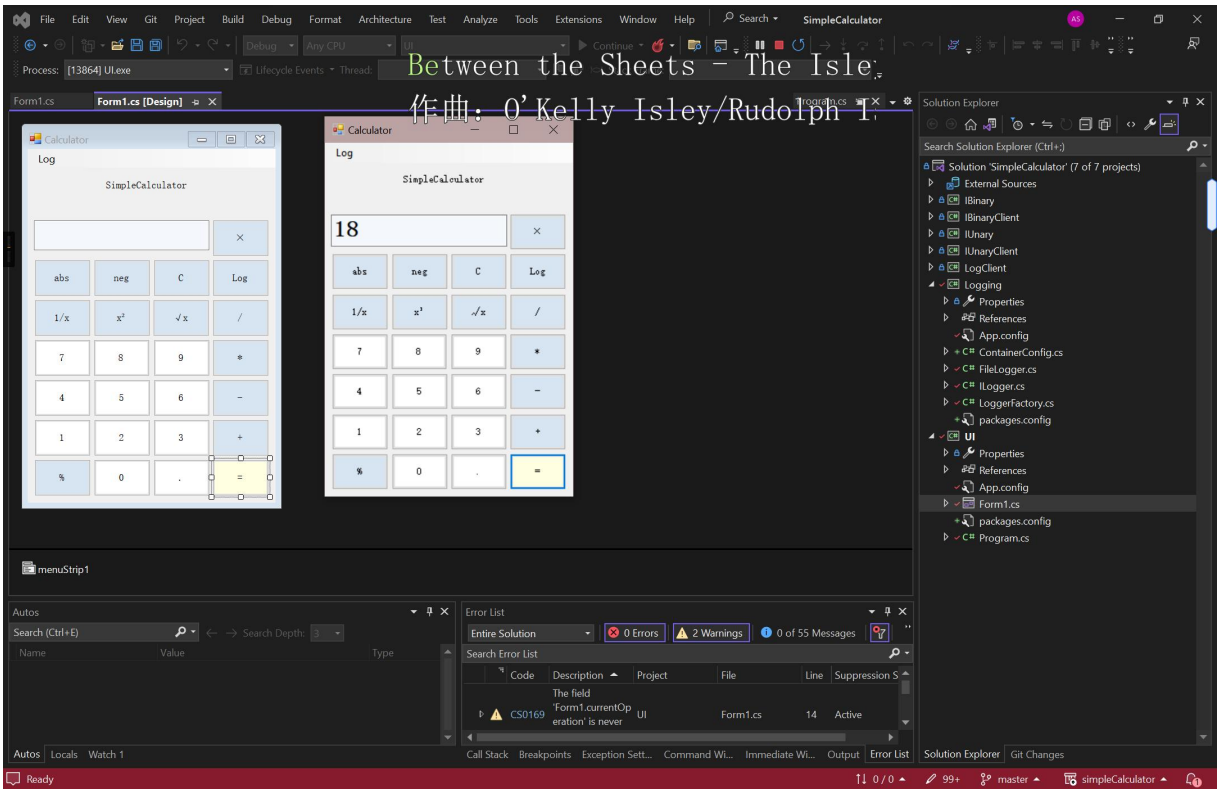

Sky-high deflating like this
Thriving of your lessons; yes,
using System;
using System.IO;
using System.Text.Json;
using System.Windows.Forms;
using Autofac;
using Logging;
namespace UI
static class Program
private static IContainer container;
[STAThread]
static void Main()
// Read settings from JSON file
var jsonString = File.ReadAllText("appsettings.json");
var config = JsonSerializer.Deserialize<Config>(jsonString);
var builder = new ContainerBuilder();
if (config.Logger.Type == "FileLogger")
builder.Register(c => new FileLogger(config.Logger.LogFilePath)).As<ILogger>();
else
throw new NotSupportedException($"Logger type '{config.Logger.Type}' is not supported.");
container = builder.Build();
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
public class Config

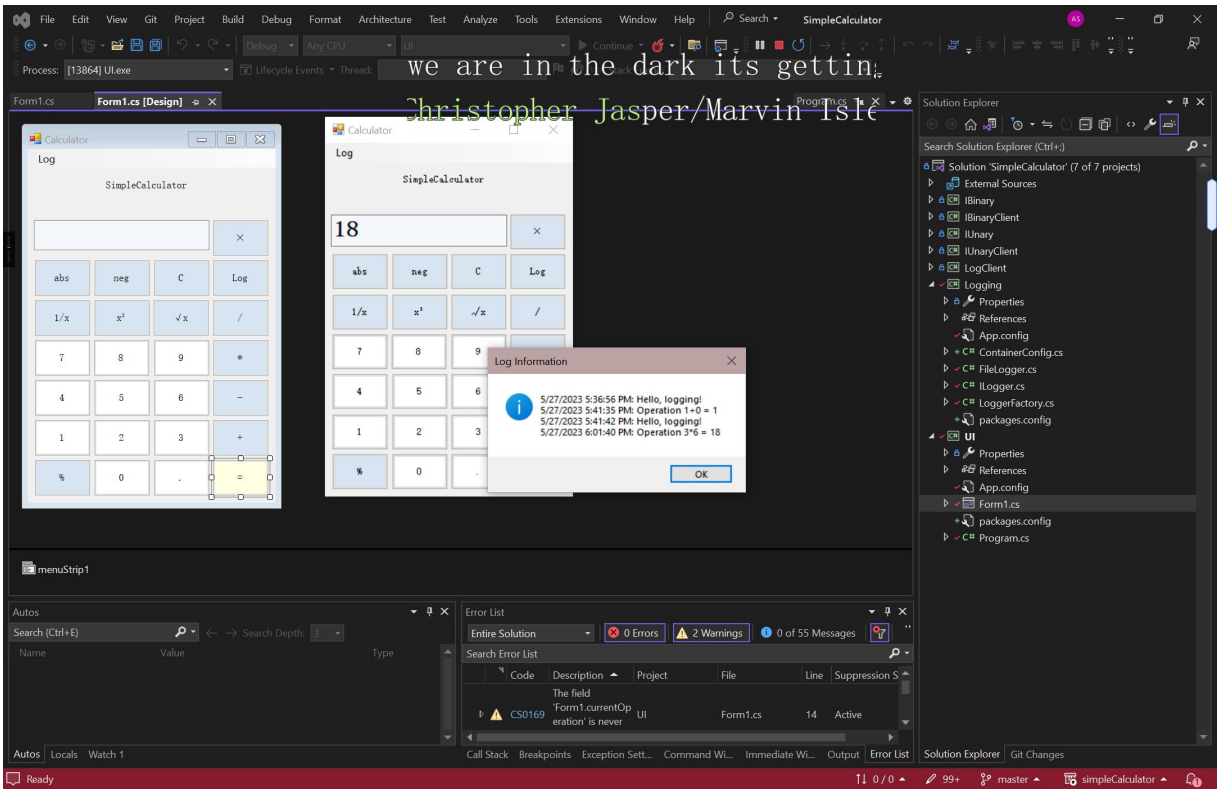


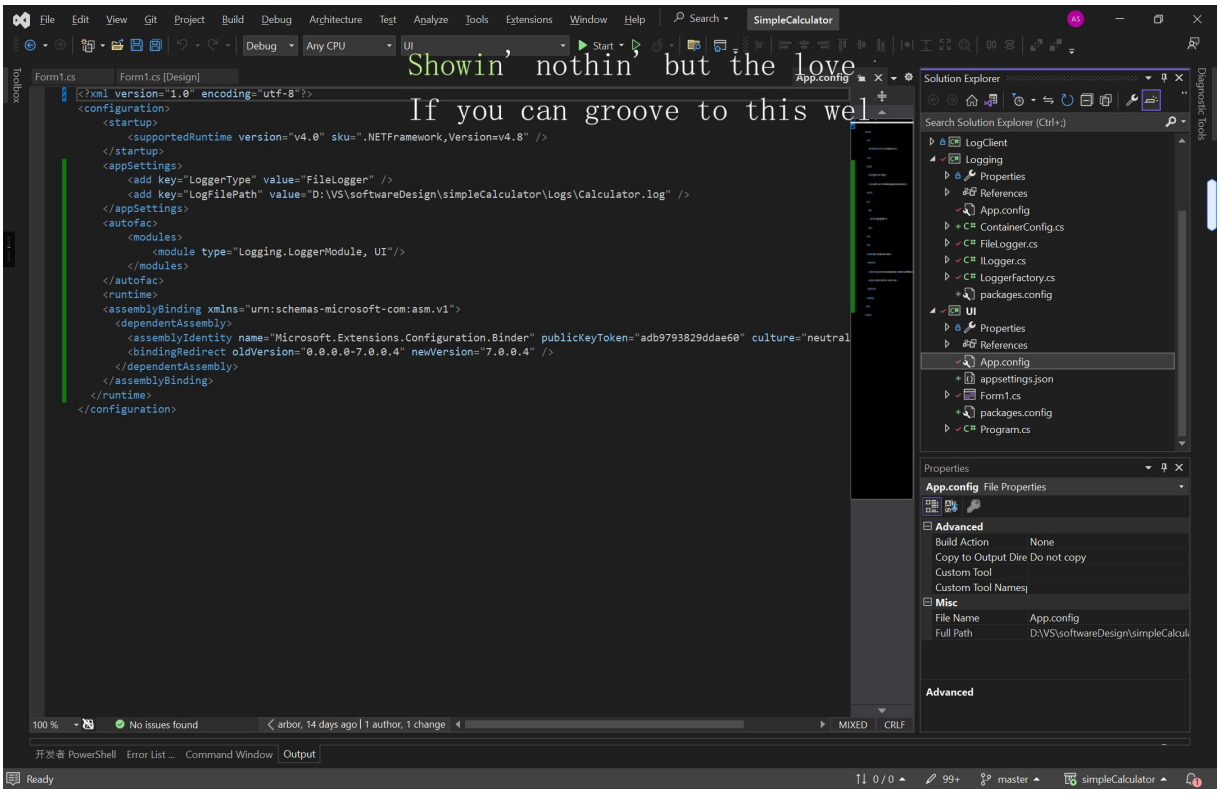
四、运行效果

（1）设计静态功能类，在静态功能类中完成组件注册

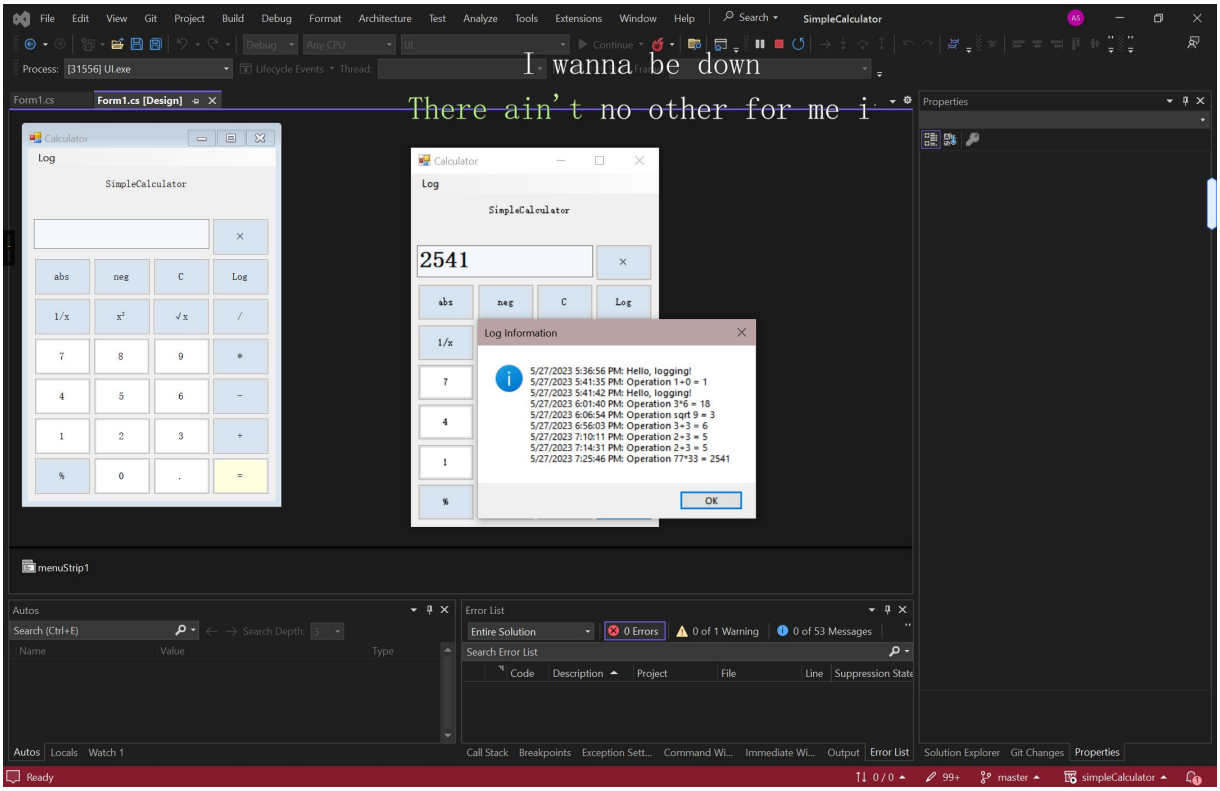



Between the Sheets – The Isley
作曲：O'Kelly Isley/Rudolph I

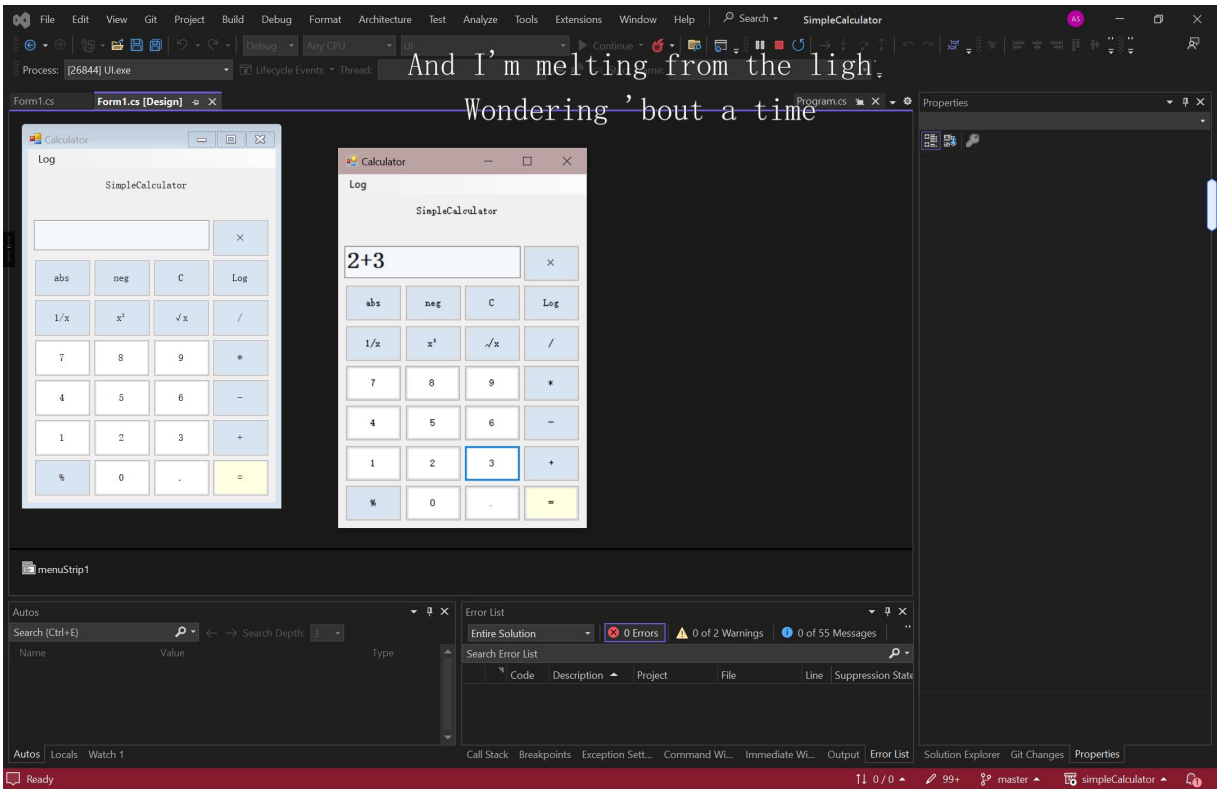

we are in the dark its gettin
Christopher Jasper/Marvin Isle

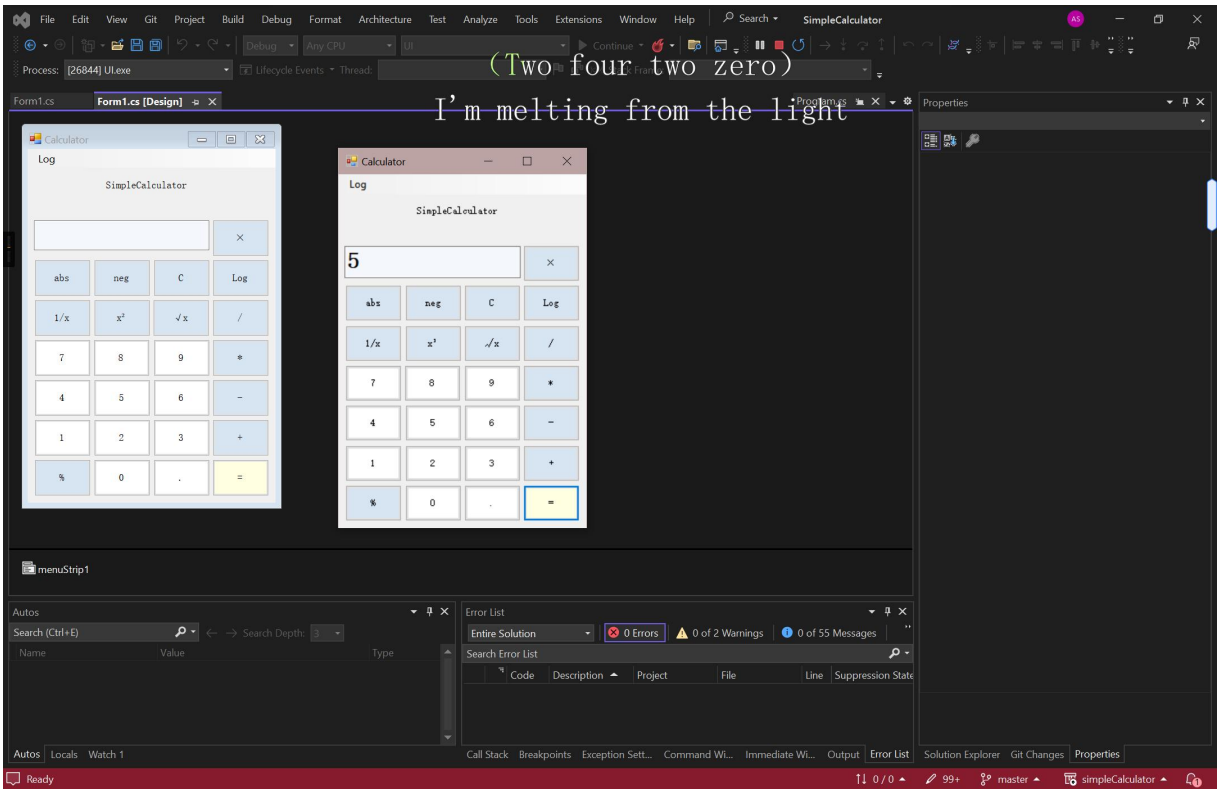


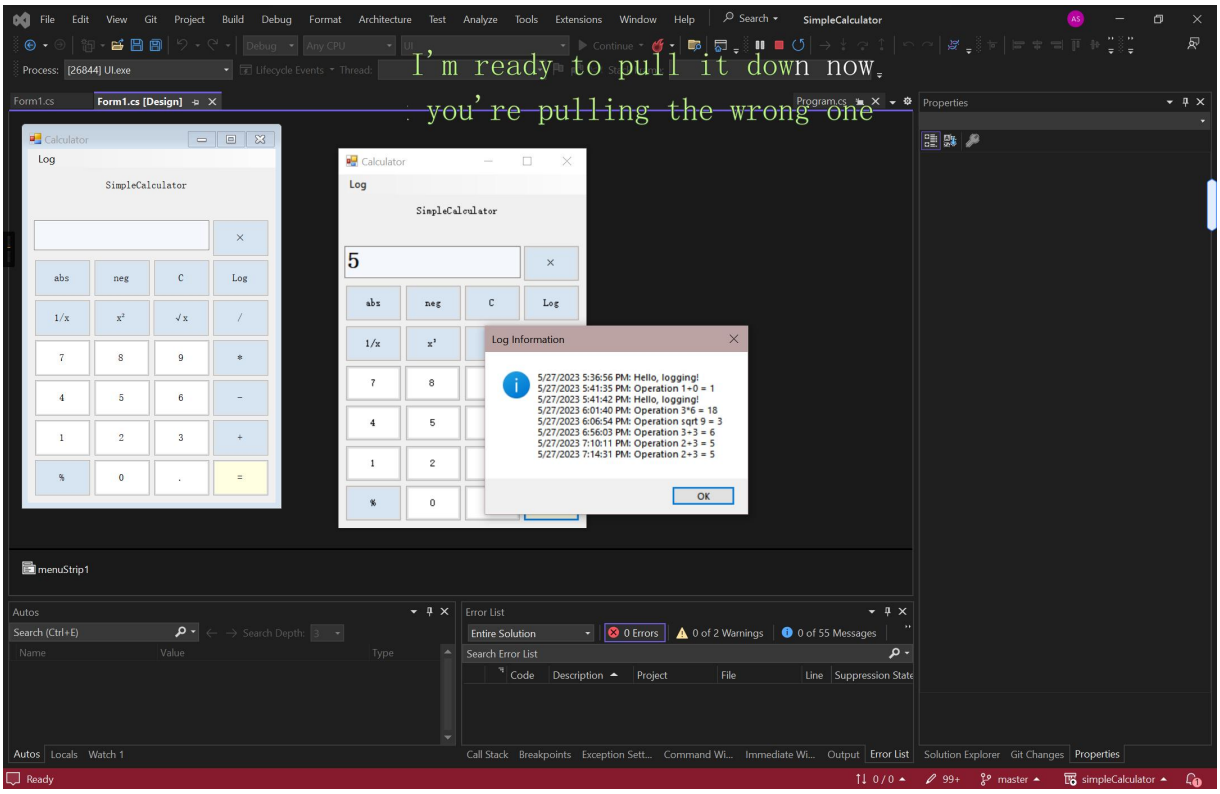
（2）使用配置文件 app.config （自定义配置节 autofac、在 autofac 中配置要注册的组件，通过 RegistModule 注册组件）



（3）使用配置文件 xx.json or xx.XML 配置要要注册的组件


And I’m melting from the ligh
Wondering ’bout a time
2+3


(Two four two zero)
I’m melting from the light
5



五、实验心得

本次实验的目标是对计算器项目进行重构，采用 IOC（Inversion of Control，控制反转）的设计思想，并使用 Autofac 框架完成日志记录器对象和窗体对象的实例化。我们采用了三种方式实现。

1. 静态功能类：首先，我们设计了静态功能类，在该类中完成了组件的注册。这使我们能够在代码中直接注册和解析依赖关系，而不需要通过外部配置。这种方法的好处是能够清晰地在代码中看到依赖关系的注册和解析，但缺点是如果需要改变依赖关系，就必须修改代码。

2. 配置文件 app.config：其次，我们使用了配置文件 app.config 完成了组件的注册，我们在 autofac 中自定义了配置节，并通过 RegistModule 注册组件。这种方法使我们能够在不修改代码的情况下改变依赖关系，提高了代码的灵活性。但是，需要注意的是，这需要对配置文件有深入的理

解，否则可能会导致错误的配置。

3.配置文件 xx.json or xx.xml：最后，我们使用了 json 或 xml 配置文件来注册组件。这种方法与使用 app.config 类似，都能提高代码的灵活性。不同的是，json 或 xml 文件比 app.config 更加灵活，能够更好地描述复杂的配置结构。不过，这种方法也需要对 json 或 xml 有一定的理解。

通过这次实验，我更深入地理解了 IOC 的设计思想，体验了如何使用 Autofac 框架进行依赖注入，并通过三种不同的方式实现了组件的注册。我认识到，虽然这三种方式各有优缺点，但是，选择哪种方式取决于具体的项目需求和团队习惯。总的来说，这次实验增强了我对 IOC 和依赖注入概念的理解，并提高了我解决实际问题的能力。