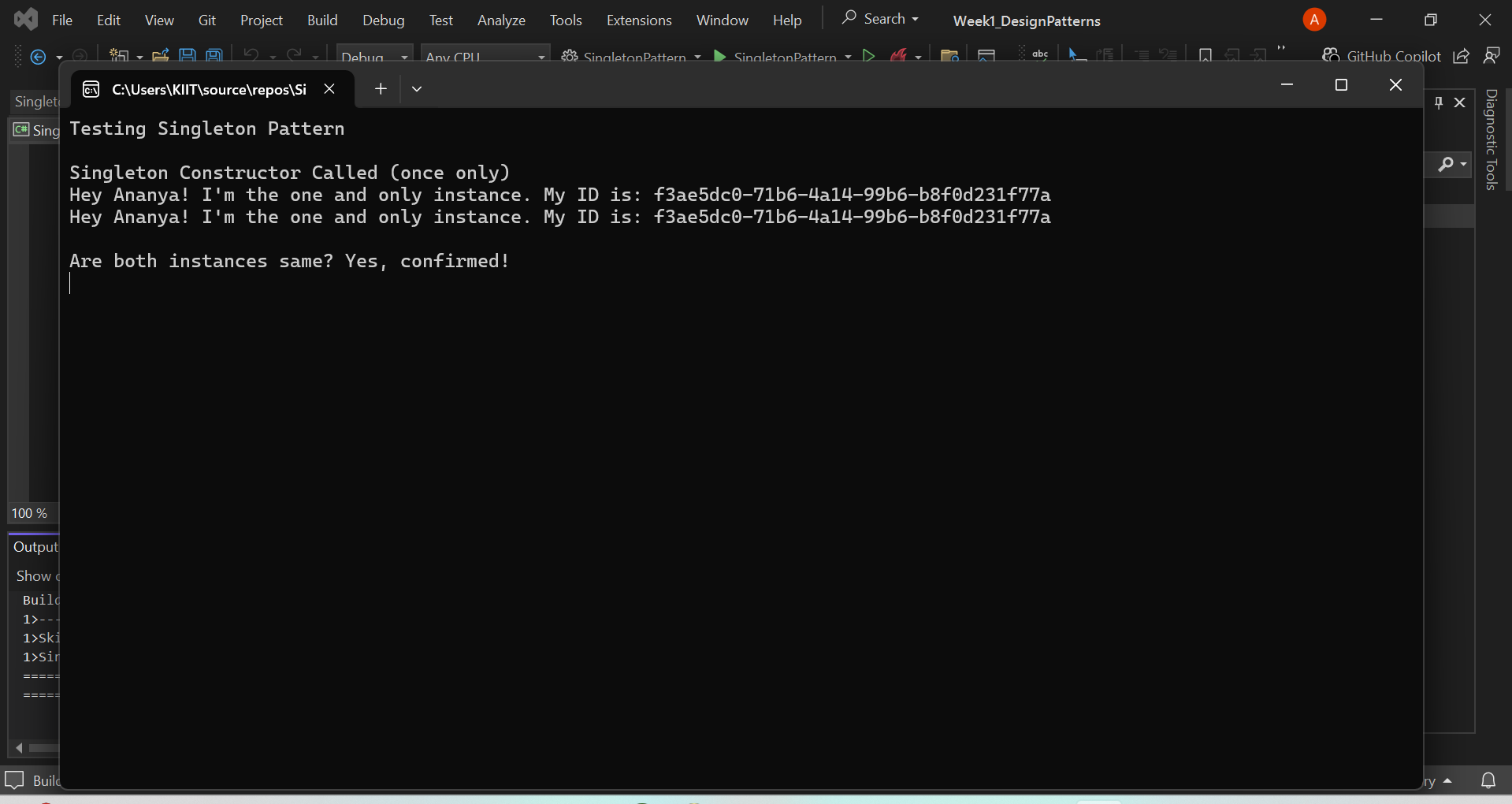
**Exercise 1: Implementing the Singleton Pattern:**

**Screenshots of OUTPUT:**

****

**CODES :**

**//program.cs**

using System;

namespace SingletonPattern

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Testing Singleton Pattern\n");

// getting instance for the first time

var instance1 = Singleton.GetInstance();

instance1.PrintMessage();

// trying again to see if same instance comes

var instance2 = Singleton.GetInstance();

instance2.PrintMessage();

Console.WriteLine($"\nAre both instances same? {(instance1 == instance2 ? "Yes, confirmed!" : "nope, smth wrong...")}");

// pause so output stays on screen

Console.ReadLine();

}

}

}

**//Singleton.cs**

using System;

namespace SingletonPattern

{

public class Singleton

{

private static Singleton \_instance;

private static readonly object \_lock = new object();

public Guid InstanceId { get; }

// private constructor bcz we dnt want outsiders to make objects

private Singleton()

{

InstanceId = Guid.NewGuid(); // random id just to see uniqueness

Console.WriteLine("Singleton Constructor Called (once only)");

}

public static Singleton GetInstance()

{

// making it thread-safe (lock for safety lol)

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Singleton(); // create only if not made yet

}

return \_instance;

}

}

public void PrintMessage()

{

Console.WriteLine($"Hey Ananya! I'm the one and only instance. My ID is: {InstanceId}");

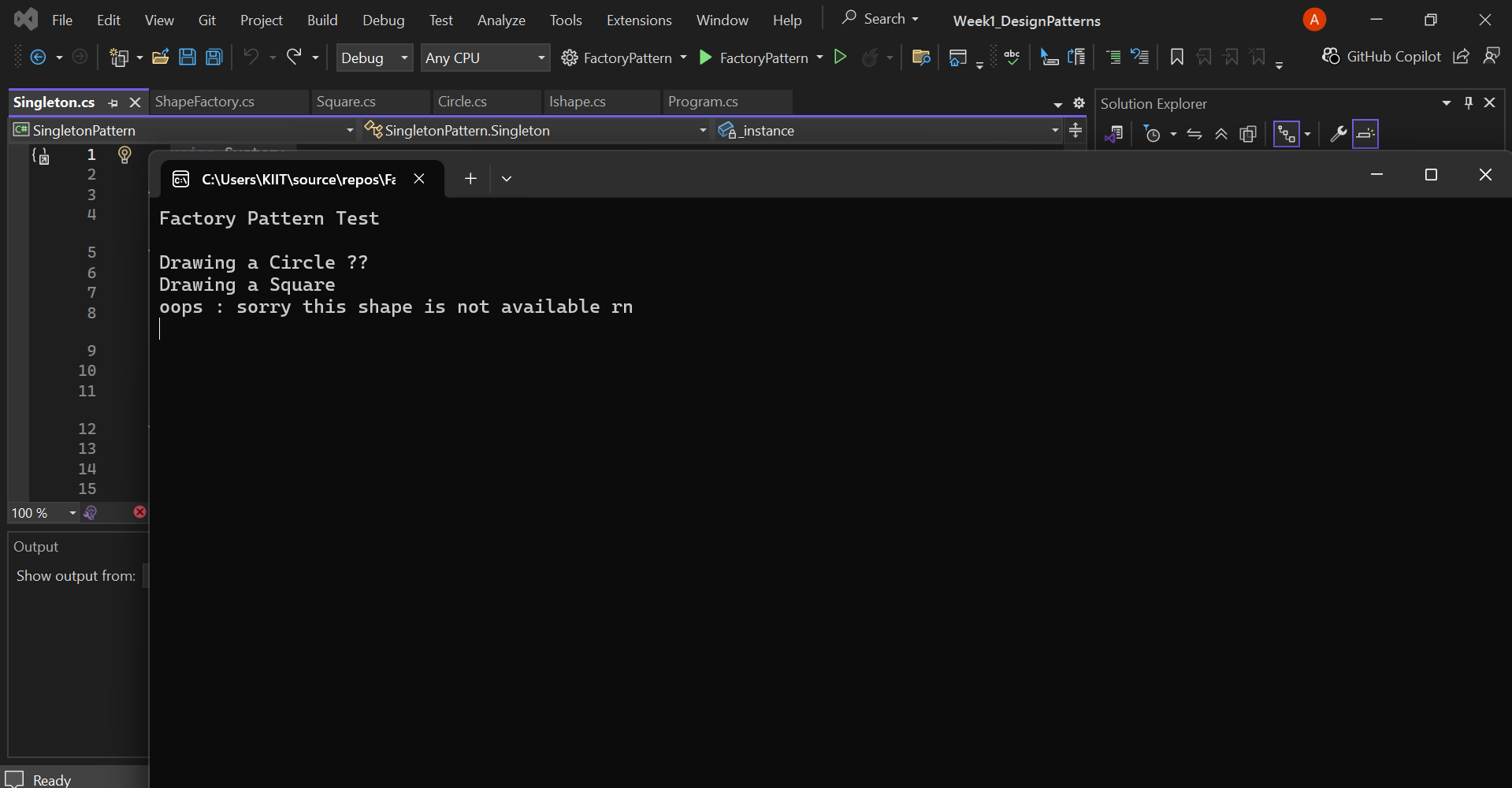
}

}

}

**Exercise 2: Implementing the Factory Method Pattern**

**Screenshots of OUTPUT:**



**CODES :**

**//program.cs**

using System;

namespace FactoryPattern

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Factory Pattern Test\n");

// using factory to create shapes

IShape shape1 = ShapeFactory.GetShape("Circle");

shape1.Draw(); // should draw circle

IShape shape2 = ShapeFactory.GetShape("Square");

shape2.Draw(); // should draw square

// testing invalid input just to see what happens

try

{

var shape3 = ShapeFactory.GetShape("Triangle");

shape3.Draw(); // shouldn't reach here

}

catch (Exception ex)

{

Console.WriteLine("oops : " + ex.Message);

}

Console.ReadLine(); // pause the output screen

}

}

}

**//Ishape.cs**

namespace FactoryPattern

{

public interface IShape

{

void Draw(); // just a draw method

}

}

**//Circle.cs**

using System;

namespace FactoryPattern

{

public class Circle : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Circle 🌕");

}

}

}

**//Square.cs**

using System;

namespace FactoryPattern

{

public class Square : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Square ");

}

}

}

**//ShapeFactory.cs**

using System;

namespace FactoryPattern

{

public class ShapeFactory

{

public static IShape GetShape(string shapeType)

{

// not case-sensitive, just using lowercase

switch (shapeType.ToLower())

{

case "circle":

return new Circle();

case "square":

return new Square();

default:

throw new ArgumentException("sorry this shape is not available rn");

}

}

}

}