

**Name**: Adeel Ali Yousaf **Class**: BS CS (Blue)

**Roll Number**: 24010102 – 011 **Semester**: 6th (Spring 2025)

**Lecturer**: Ma’am Hina Farooq **Subject**: Visual Programming

**Assignment no.2**

**Question no.1:**

**Student Grades Classification Create a Student class with properties Name, RollNumber, and Marks. Implement a method GetGrade() that returns the grade based on marks:**

**90 and above: A**

**80-89: B**

**70-79: C**

**Below 70: Fail**

**In the Main() method, create multiple student objects and display their grades.**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

class Student

{

private string Name;

private string RollNumber;

private int Marks;

public Student(string Name, string RollNumber, int Marks)

{

this.Name = Name;

this.RollNumber = RollNumber;

this.Marks = Marks;

getGrade();

}

private void getGrade()

{

if(Marks >= 90)

{

Console.WriteLine($"Congratulations! {Name}, You Have Acheived Grade: A");

}

else if(Marks>=80 && Marks <= 89)

{

Console.WriteLine($"Congratulations! {Name}, You Have Acheived Grade: B");

}

else if (Marks >= 70 && Marks <= 79)

{

Console.WriteLine($"Congratulations! {Name}, You Have Acheived Grade: C");

}

else if (Marks < 70)

{

Console.WriteLine($"Better Luck Next Time! {Name}, You Have Been Failed.");

}

}

}

internal class Program

{

static void Main()

{

Student s1 = new Student("Adeel", "24010102-011", 95);

Student s2 = new Student("Ali", "24010102-012", 85);

Student s3 = new Student("Yousaf", "24010102-013", 75);

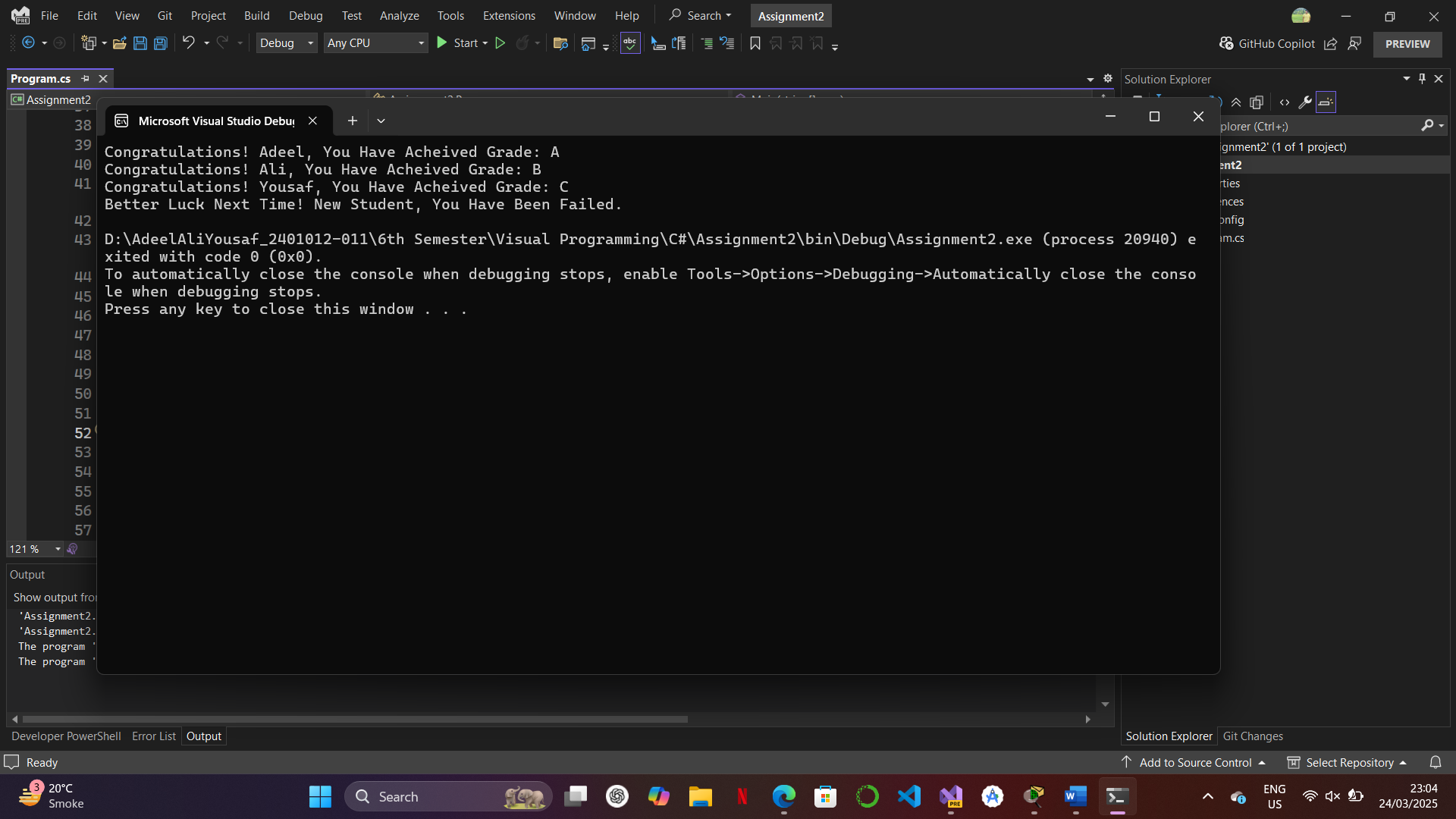
Student s4 = new Student("New Student", "24010102-00", 65);

}

}

}

* **Output**:



**Question no.2:**

**Bank Account Transactions Create a base class BankAccount with properties AccountNumber, Balance, and methods Deposit() and Withdraw(). Create a derived class SavingsAccount that adds an interest rate property and a method ApplyInterest(). Ensure that withdrawal does not allow negative balance. Demonstrate depositing, withdrawing, and applying interest using objects.**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

class BankAccount

{

public string AccountNumber;

public double Balance = 0;

public BankAccount(string accountNumber)

{

this.AccountNumber = accountNumber;

Console.WriteLine($"Account Number: {accountNumber}");

}

public void Desposit()

{

Console.WriteLine("This is a Desposit Method. Enter an Amount to Deposit: ");

double depositAmount = Convert.ToDouble(Console.ReadLine());

Balance = depositAmount + Balance;

Console.WriteLine($"\nYour Current Balance has been Updated! Which is: {Balance}pkr.");

}

public void Withdraw()

{

Console.WriteLine("\nThis is a Withdraw Method. Enter an Amount to Withdraw: ");

double withdrawAmount = Convert.ToDouble(Console.ReadLine());

if(withdrawAmount > Balance)

{

Console.WriteLine($"\nSorry! Your Withdraw amount: {withdrawAmount}pkr is Greater than Your Current Balance which is: {Balance}pkr");

}

else

{

Balance = Balance - withdrawAmount;

Console.WriteLine($"\nYour Current Balance has been Updated! Which is: {Balance}pkr\n");

}

}

}

class SavingAccount:BankAccount

{

public double interestRate;

public SavingAccount(string accountNumber,double InterestRate):base(accountNumber)

{

this.interestRate = InterestRate;

}

public void ApplyInterest()

{

double temp;

temp = Balance \* interestRate / 100;

Balance = Balance - temp;

if(temp > Balance)

{

Console.WriteLine($"\nTransaction Declined Not Enough Balance. Current Balance is: {Balance}pkr and Interest Amount is: {temp}pkr.");

Console.WriteLine();

}

else

{

Console.WriteLine($"\nInterest Rate of {interestRate}% has been deducted from your account which is: {temp}pkr Your Current Balance is: {Balance}pkr.");

Console.WriteLine() ;

}

}

}

internal class Program2

{

static void Main()

{

SavingAccount ac1 = new SavingAccount("24010102-011", 17.12);

ac1.Desposit();

ac1.Withdraw();

ac1.ApplyInterest();

SavingAccount ac2 = new SavingAccount("24010102-012", 5.65);

ac2.Desposit();

ac2.Withdraw();

SavingAccount ac3 = new SavingAccount("24010102-013", 12.59);

ac3.Withdraw();

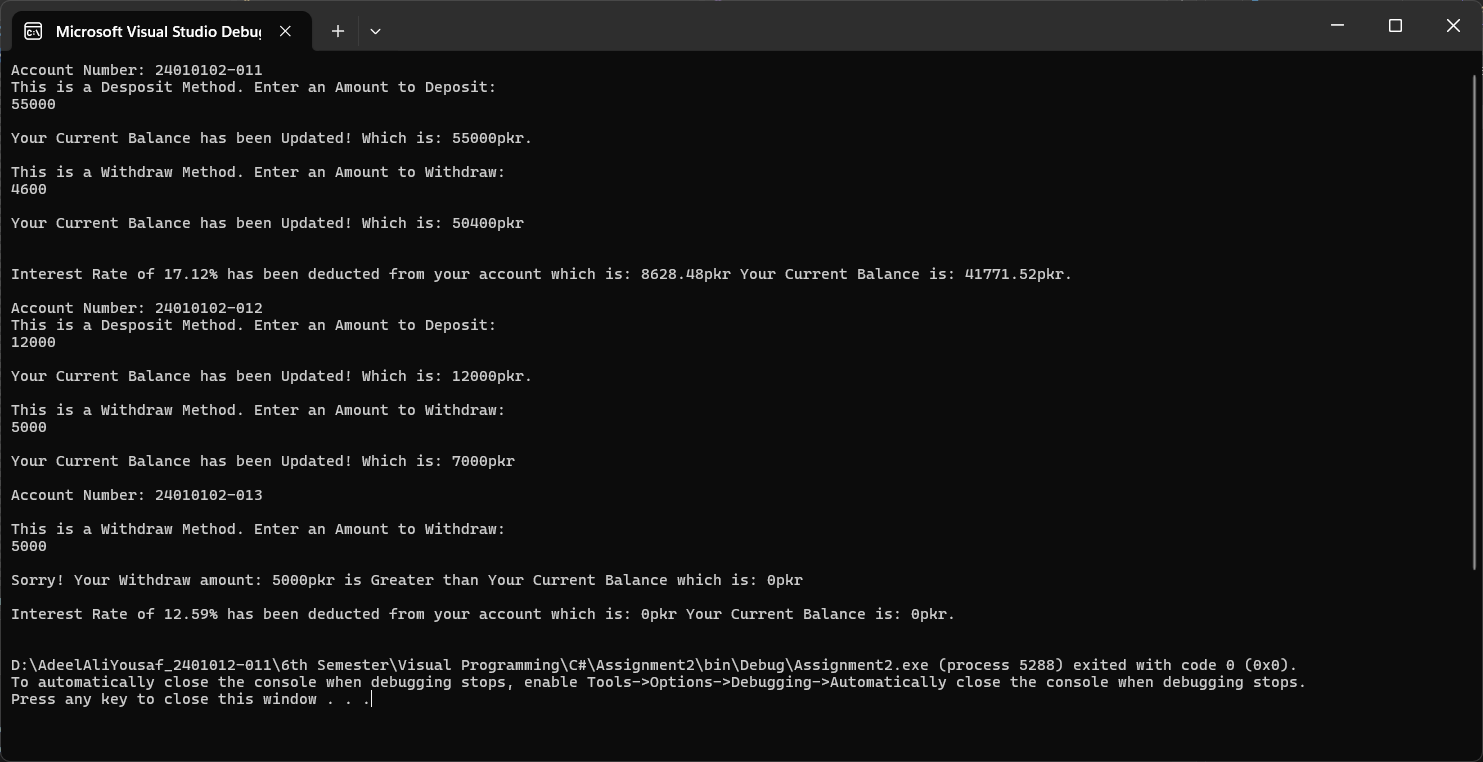
ac3.ApplyInterest();

}

}

}

* **Output**:



**Question no.3:**

**Employee Salary Calculation Implement a base class Employee with properties Name and BasicSalary. Create a derived class Manager that adds a Bonus property and overrides a method CalculateSalary() to include bonus. Also, create a Developer class that includes a ProjectAllowance and overrides CalculateSalary(). Use polymorphism to display the salary details of both employee types.**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

class Employee

{

protected string Name;

protected int BasicSalary;

public Employee(string name, int salary)

{

this.Name = name;

this.BasicSalary = salary;

}

public virtual void CalculateSalary()

{

Console.WriteLine($"Basic Salary is: {BasicSalary}pkr");

}

}

class Manager:Employee

{

protected int Bonus;

public Manager(int bonus, string name, int salary):base(name, salary)

{

this.Bonus = bonus;

}

public override void CalculateSalary()

{

int totalSalary = BasicSalary + Bonus;

Console.WriteLine($"Employee Name:{Name}, Bonus {Bonus}pkr, Total Salary is: {totalSalary}pkr, Basic Salary is: {BasicSalary}pkr.");

}

}

class Developer:Employee

{

private int ProjectAllowance;

public Developer(int projectAllowance,string name, int salary):base( name, salary)

{

this.ProjectAllowance = projectAllowance;

}

public override void CalculateSalary()

{

int totalSalary = BasicSalary + ProjectAllowance;

Console.WriteLine($"\nEmployee Name:{Name}, Project Allowance: {ProjectAllowance}pkr, Total Salary is: {totalSalary}pkr, Basic Salary is: {BasicSalary}pkr.");

} }

internal class Program3

{

static void Main()

{

Manager m1 = new Manager(23000, "Muhammad Yousaf", 75000);

m1.CalculateSalary();

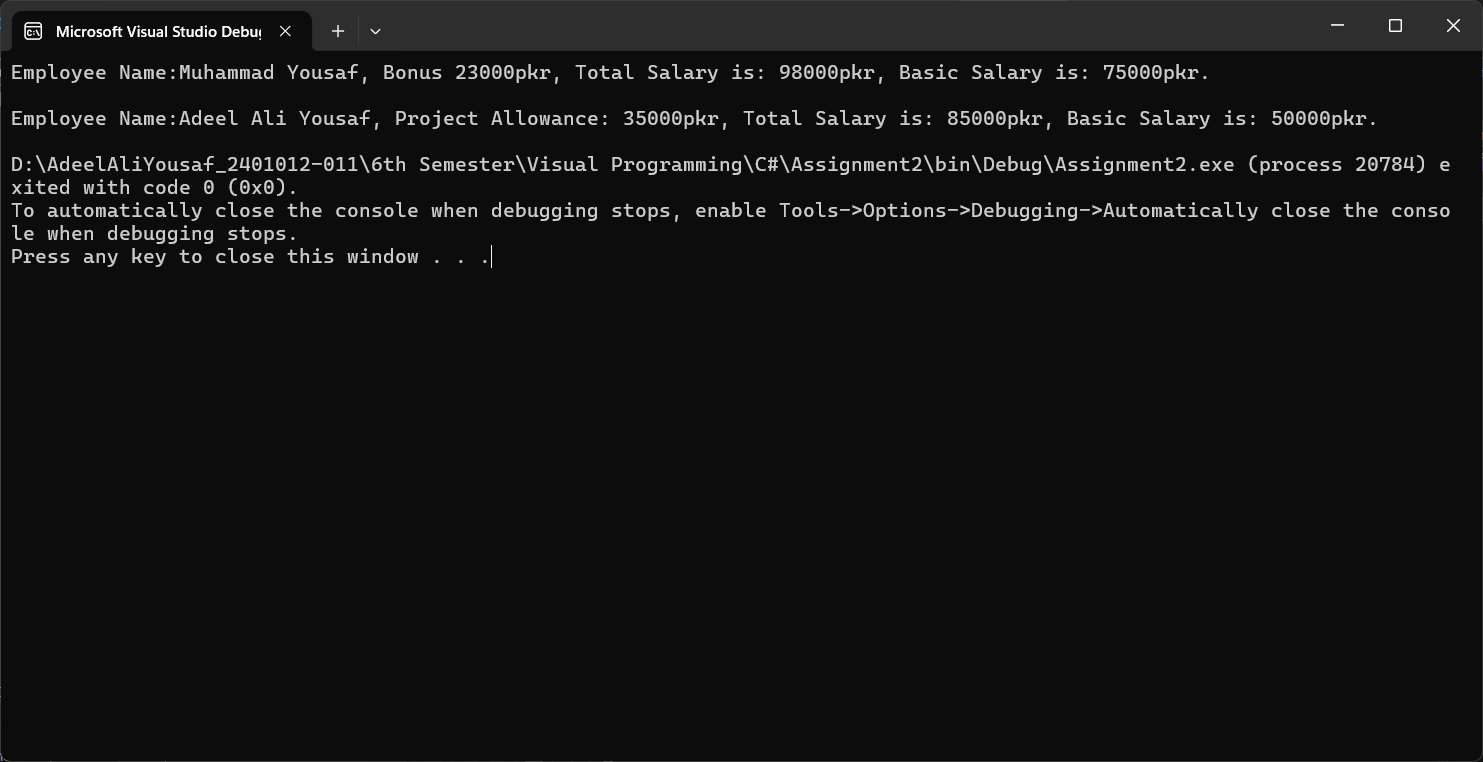
Developer d1 = new Developer(35000, "Adeel Ali Yousaf", 50000);

d1.CalculateSalary();

}

} }

* **Output**:



**Question no.4:  
Create an abstract class Shape with an abstract method CalculateArea(). Derive two classes, Rectangle and Circle, implementing the CalculateArea() method. The Rectangle class should take Length and Width, and the Circle class should take Radius. Demonstrate polymorphism by calling CalculateArea() through a Shape reference.**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

abstract class Shape

{

public abstract void CalculateArea();

}

class Rectangle: Shape

{

public override void CalculateArea()

{

Console.WriteLine("This is a Rectangle Class. Enter Length and Width to Calculate Area: ");

double length = Convert.ToDouble(Console.ReadLine());

double width = Convert.ToDouble(Console.ReadLine());

double area = length \* width;

Console.WriteLine($"Area of Rectangle is: {area}\n");

}

}

class Circle: Shape

{

public override void CalculateArea()

{

Console.WriteLine("This is a Circle Class. Enter Radius to Calculate Area: ");

double radius = Convert.ToDouble(Console.ReadLine());

double area = Math.PI \* radius \* radius;

Console.WriteLine($"Area of Circle is: {area}");

}

}

internal class Program4

{

static void Main()

{

Shape rect = new Rectangle();

Shape circle = new Circle();

rect.CalculateArea();

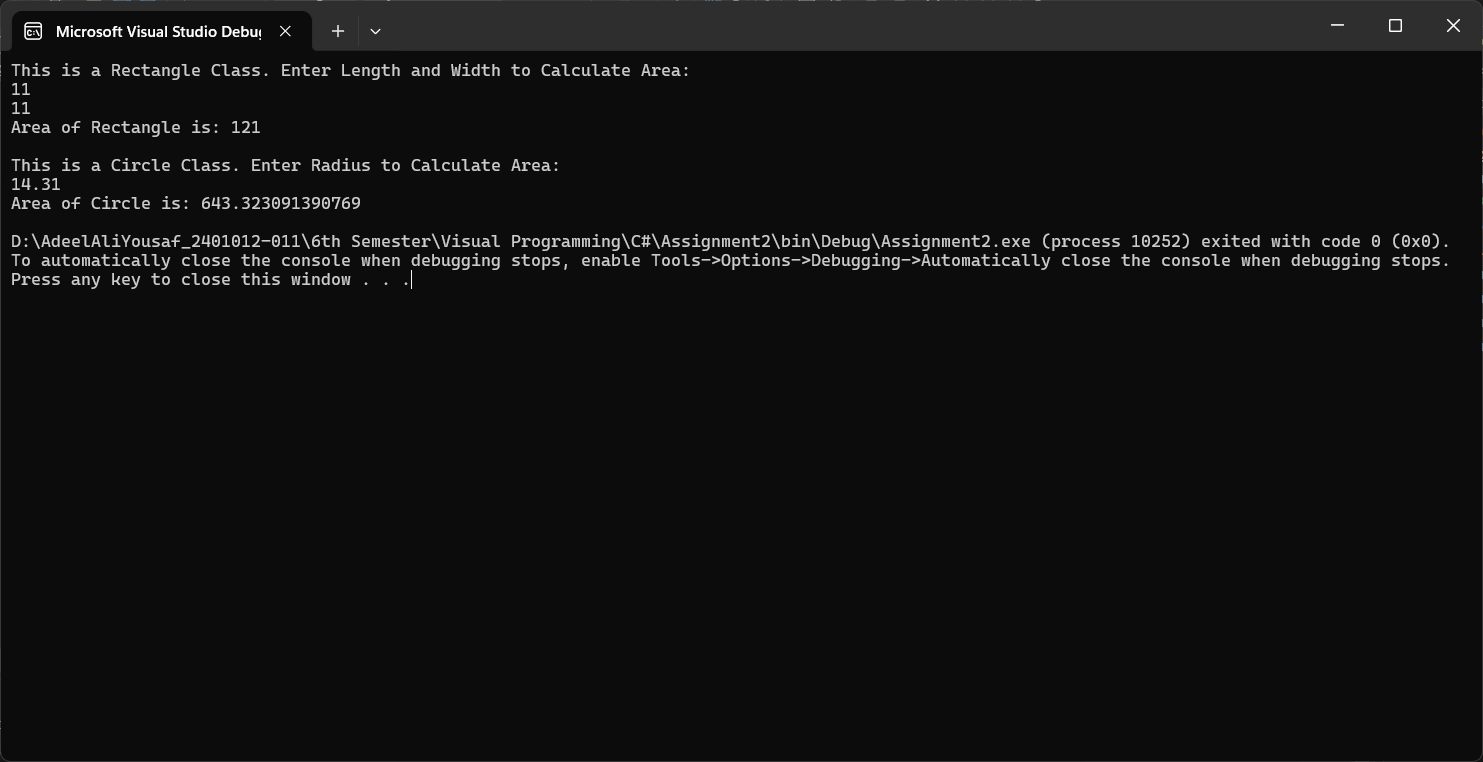
circle.CalculateArea();

}

}

}

* **Output**:



**Question no.5:**

**Create a class Calculator with overloaded methods Compute() to:**

**Add two integers**

**Multiply two doubles**

**Concatenate two strings**

**Now, create a derived class ScientificCalculator that overrides Compute() to calculate the power of a number (a^b). Demonstrate early binding (overloading) and late binding (overriding)**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

class Calculator

{

public void Compute(int a, int b)

{

Console.WriteLine($"Addition of 2 integers: {a} and {b} is: {a + b}");

}

public void Compute(double a, double b)

{

Console.WriteLine($"Multiplication of 2 doubles: {a} and {b} is: {a \* b}");

}

public void Compute(string a, string b)

{

Console.WriteLine($"Concatenation of 2 strings: {a} and {b} is: {a + b}");

}

public virtual void Compute(double a, int b)

{

Console.WriteLine("This is a Virtual Method in Base Class");

}

}

class ScientificCalculator : Calculator

{

public override void Compute(double a, int b)

{

double result = 1;

for (int i = 0; i < b; i++)

{

result \*= a;

}

Console.WriteLine($"This is a Override Method: Power of {a} to {b} is: {result}");

}

}

internal class Program5

{

static void Main()

{

Calculator calc = new Calculator();

calc.Compute(5, 10);

calc.Compute(5.5, 10.5);

calc.Compute("Adeel Ali Yousaf - ", "24010102-011");

ScientificCalculator sciCalc = new ScientificCalculator();

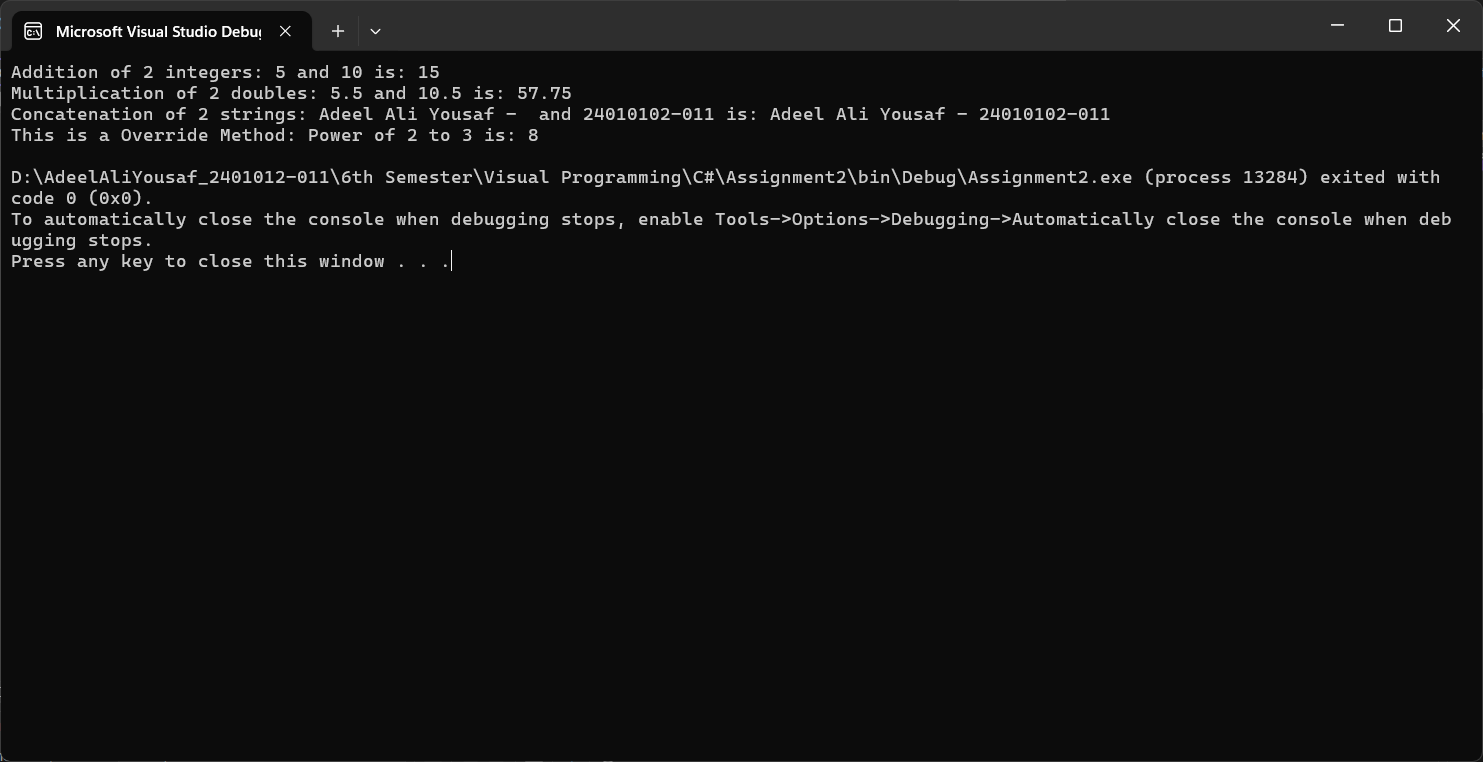
sciCalc.Compute(2.0, 3);

}

}

}

* **Output**:



**Question no.6:**

**Create a class Vehicle with properties Make, Model, and a method Start(). Create a derived class Car that overrides Start(). Now, create a sealed class ElectricCar that extends Car and implements ChargeBattery(). Try to create a derived class from ElectricCar and observe the compilation error.**

**Answer:**

* **Code:**

using System;

namespace Assignment2

{

class Vehicle

{

public string Make;

public string Model;

public virtual void Start()

{

Console.WriteLine("This is a virtual method in Vehicle class.");

}

}

class Car: Vehicle

{

public override void Start()

{

Console.WriteLine("This is a override method in Car class.");

}

}

sealed class ElectricCar: Car

{

public void ChargeBattery()

{

Console.WriteLine("This is a ChargeBattery() in Electric Car class.");

}

}

class Tesla : ElectricCar

{

public void Autopilot()

{

Console.WriteLine("This is a Autopilot() in Tesla class.");

} }

internal class Program6

{

static void Main()

{

Vehicle vehicle = new Vehicle();

vehicle.Start();

Car car = new Car();

car.Start();

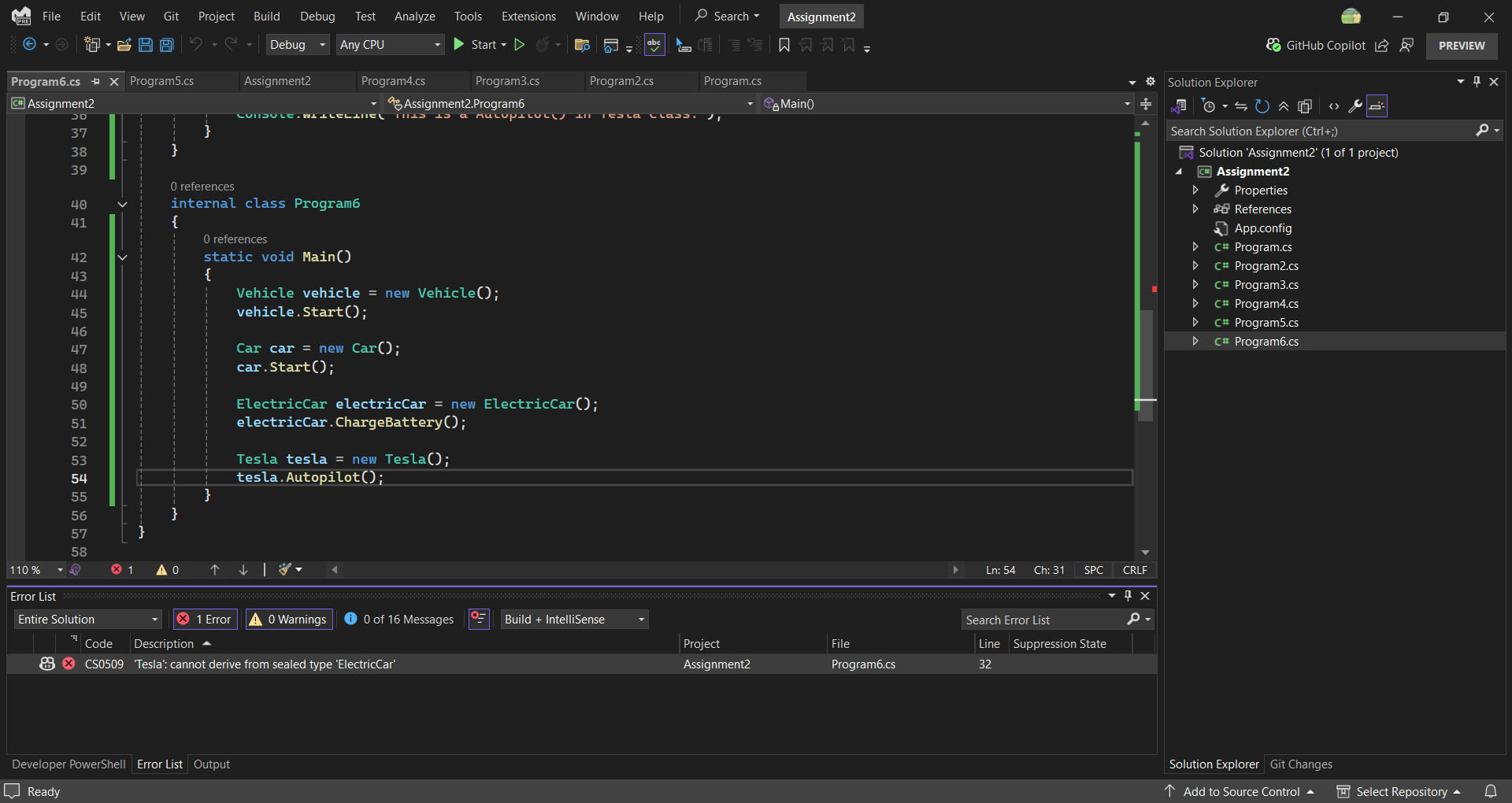
ElectricCar electricCar = new ElectricCar();

electricCar.ChargeBattery();

Tesla tesla = new Tesla();

tesla.Autopilot(); } } }

* **Output**:



* **Compilation Error**:

A compilation error was occurd because “ElectricCar” was sealed according to the question. Which does not allow inheritance. So the error was generated “ 'Tesla': cannot derive from sealed type 'ElectricCar' ”. To solve this error we should remove the “sealed” keyword from the Base Class which is ElectricCar so the derived class which is Tesla can inherit its properties.