**Q1. Handling Division by Zero.**

Two numbers and perform division. Use try-catch-finally. Catch DivideByZeroException and display “Division by zero is not allowed.” In the finally block display “Execution completed.” Ensure finally executes regardless of exceptions.

**Code:**

using System;

class Program {

static void Main() {

Console.Write("Enter first number: ");

int num1 = int.Parse(Console.ReadLine());

Console.Write("Enter second number: ");

int num2 = int.Parse(Console.ReadLine());

try {

int result = num1 / num2;

Console.WriteLine("Result: " + result);

}

catch (DivideByZeroException) {

Console.WriteLine("Division by zero is not allowed."); {

finally

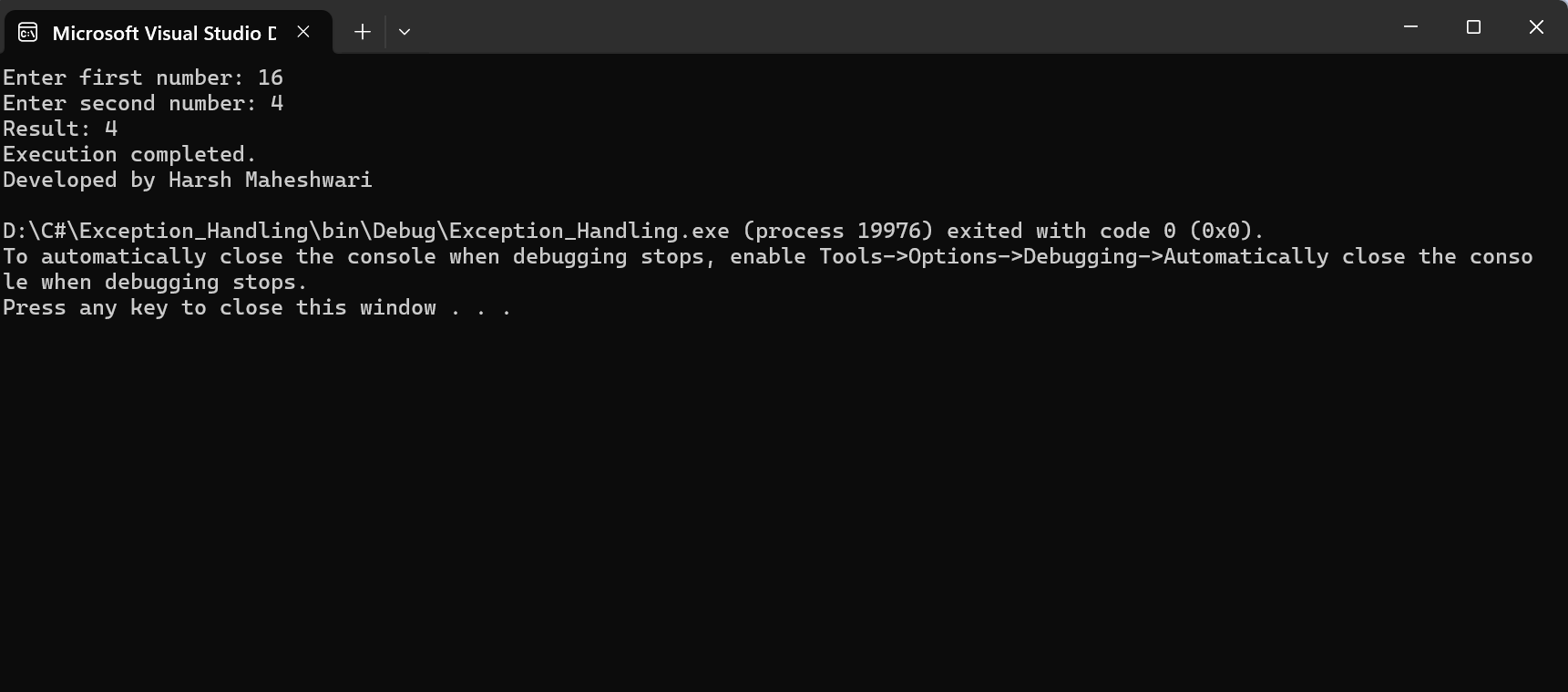
Console.WriteLine("Execution completed.");

}

}

}

**Output:**



**Q2. Multiple Catch Blocks.**

Read console input and convert to int. Handle FormatException, OverflowException, and a generic Exception, with distinct messages.

using System;

**Code:**

class Program {

static void Main() {

Console.Write("Enter a number: ");

string input = Console.ReadLine();

try {

int number = Convert.ToInt32(input);

Console.WriteLine("You entered: " + number);

catch (FormatException) }

{ Console.WriteLine("Input is not a valid number.");

}

catch (OverflowException) {

Console.WriteLine("Number is too large or too small.");

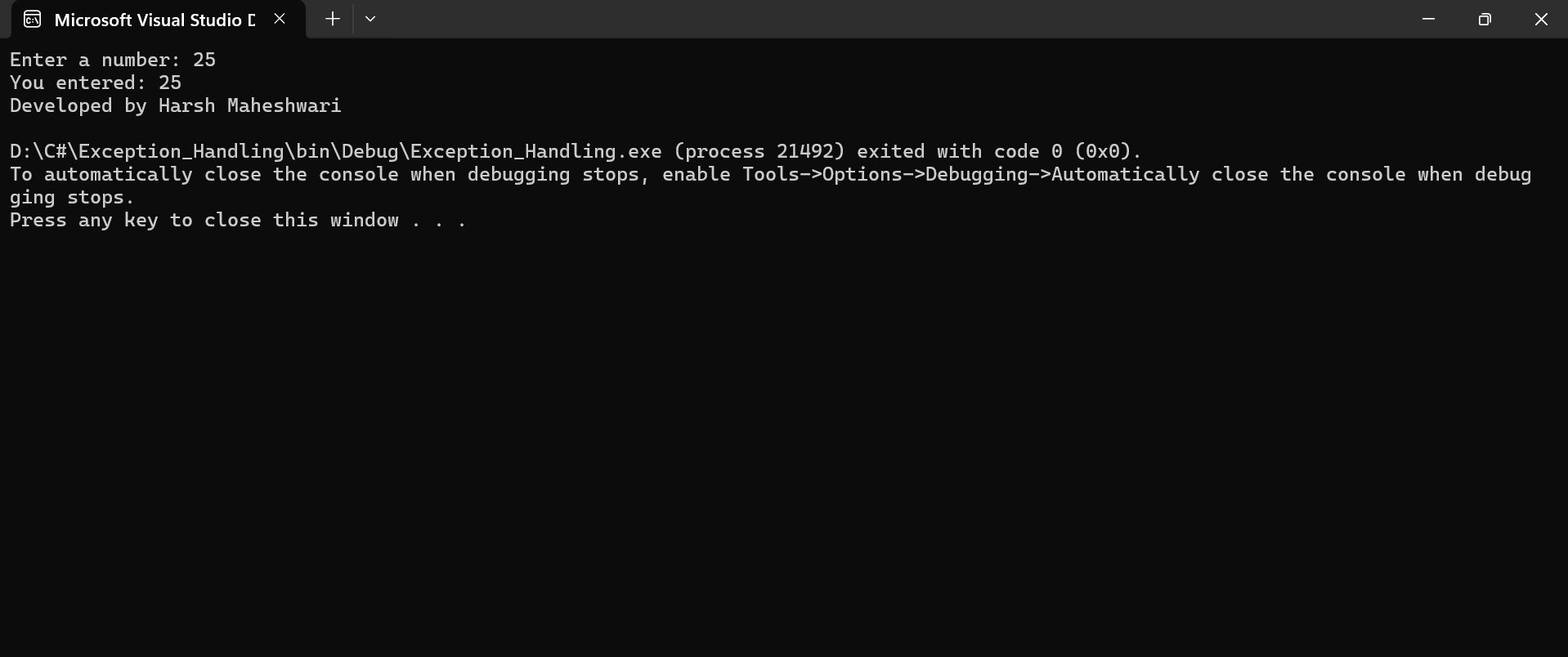
}

catch (Exception) {

Console.WriteLine("An unexpected error occurred.");

} } }

**Output:**



**Q3. Custom Exception — NegativeSalaryException**.

Define NegativeSalaryException : Exception. If entered salary < 0, throw it and handle with a clear error message.

using System;

class NegativeSalaryException : Exception {

public NegativeSalaryException(string message) : base(message) { }

}

class Program {

static void Main() {

Console.Write("Enter salary: ");

double salary = double.Parse(Console.ReadLine());

try {

if (salary < 0)

throw new NegativeSalaryException("Salary cannot be negative.");

Console.WriteLine("Salary entered: " + salary);

}

catch (NegativeSalaryException ex) {

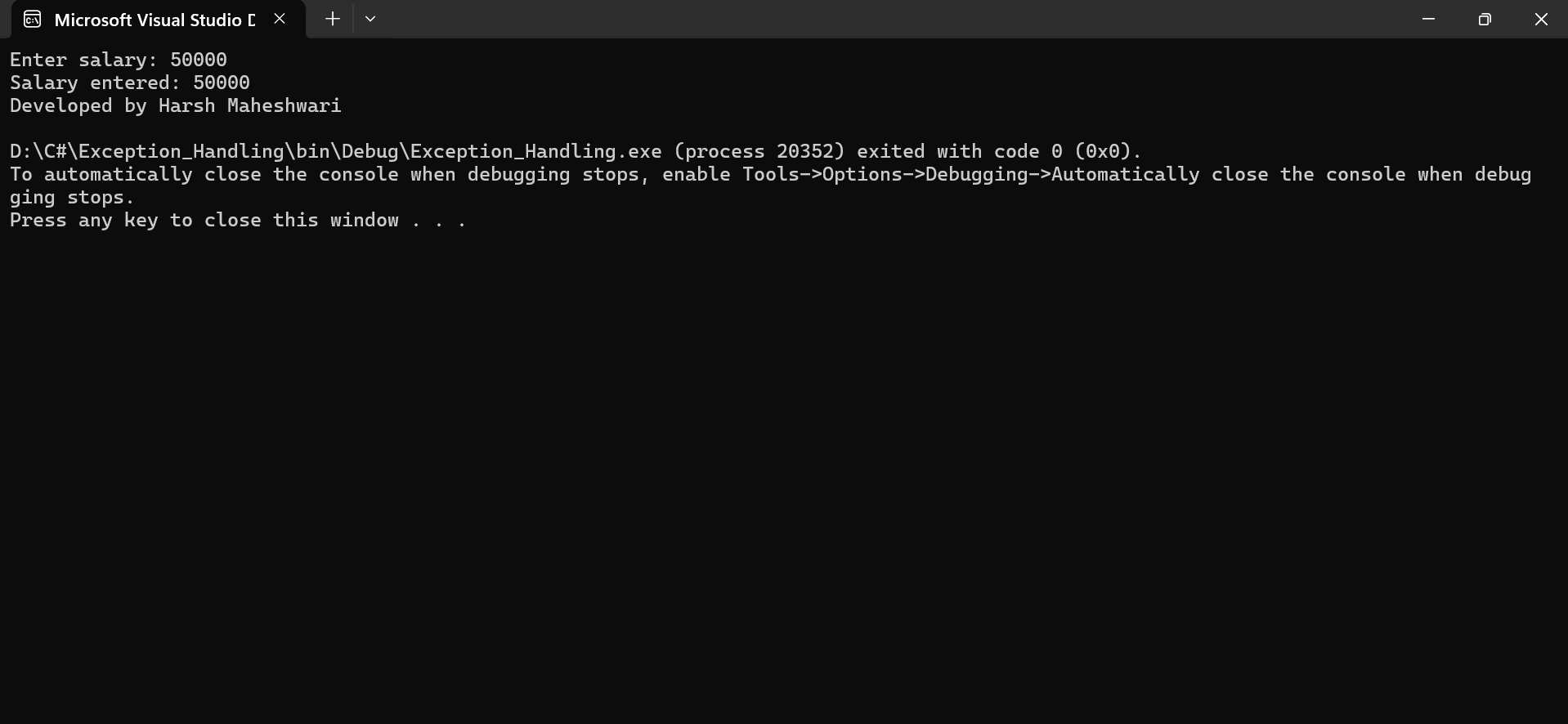
Console.WriteLine(ex.Message);

}

Console.WriteLine("Developed by Harsh Maheshwari");

} }

**Output:**

****

**Q4. Banking Scenario — InsufficientBalanceException.**

Simulate withdrawal: if withdrawal > balance, throw custom InsufficientBalanceException; otherwise print remaining balance.

using System;

class InsufficientBalanceException : Exception {

public InsufficientBalanceException(string message) : base(message) { }

}

class Program {

static void Main() {

double balance = 5000;

Console.Write("Enter withdrawal amount: ");

double withdrawal = double.Parse(Console.ReadLine());

try {

if (withdrawal > balance)

throw new InsufficientBalanceException("Insufficient balance for withdrawal.");

balance -= withdrawal;

Console.WriteLine("Withdrawal successful. Remaining balance: " + balance);

}

catch (InsufficientBalanceException ex) {

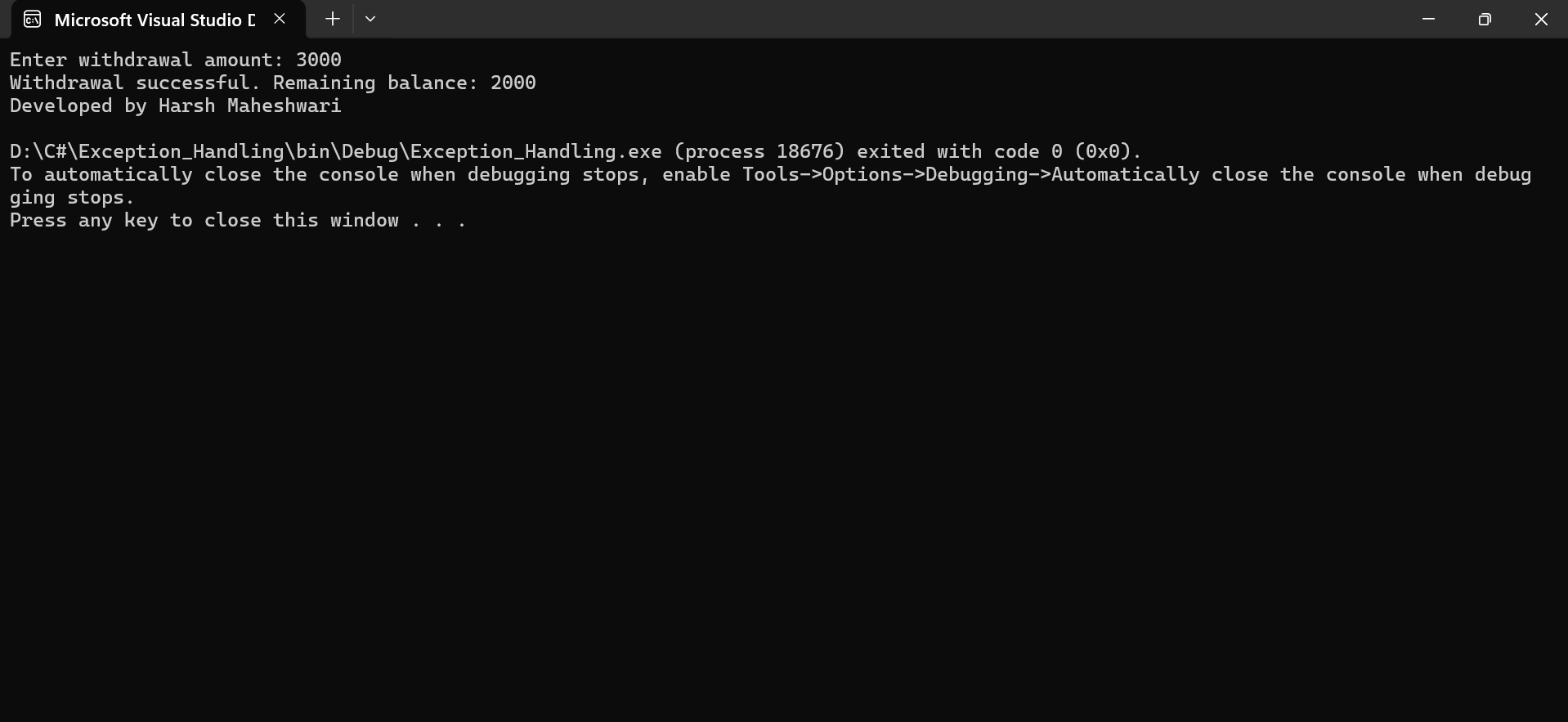
Console.WriteLine(ex.Message);

}

Console.WriteLine("Developed by Harsh Maheshwari");

} }

**Output:**

****

**Q5. Student Marks Validation.**

**Code:**

Student class stores marks (0–100). If input outside range, throw InvalidMarksException. Demonstrate validation and handling in Main().

using System;

class InvalidMarksException : Exception {

public InvalidMarksException(string message) : base(message) { } }

class Student {

private int marks;

public void SetMarks(int m) {

if (m < 0 || m > 100)

throw new InvalidMarksException("Marks should be between 0 and 100.");

marks = m;

}

public int GetMarks() {

return marks; } }

class Program {

static void Main() {

Student s = new Student();

Console.Write("Enter student marks: ");

int inputMarks = int.Parse(Console.ReadLine());

try {

s.SetMarks(inputMarks);

Console.WriteLine("Marks recorded: " + s.GetMarks());

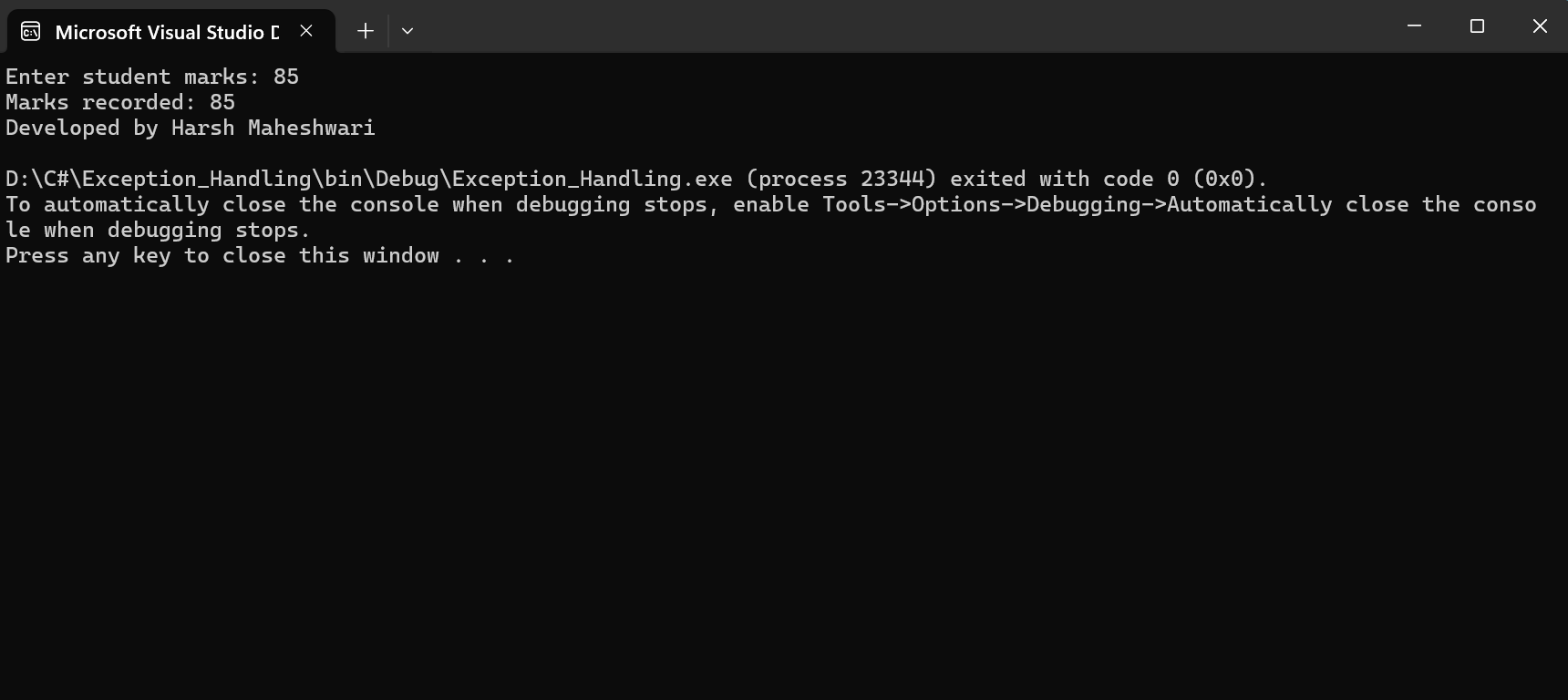
}

catch (InvalidMarksException ex) {

Console.WriteLine(ex.Message);

} } }

**Output:**

****

**MCQ Questions:**

1. **Which of the following keywords is used to handle exceptions in C#?**

Ans: try.

1. **What does the finally block do in C#?**

Ans: Executes always, whether exception occurs or not.

1. **Which class is the base for all exceptions in C#?**

Ans: Exception

1. What happens if an exception is not handled in any method?

Ans: The program terminates abnormally.

1. Which statement is used to manually raise an exception?

Ans: throw

1. What will be the output of dividing by zero in C#?

Ans: DivideByZeroException

1. Which of the following is true about multiple catch blocks?

Ans: More specific exceptions must appear before general ones

1. Can a finally block be used without a catch block?

Ans: Yes.

1. Predict the output

using System;

class Test{

static void Main() {

try {

int x = 10, y = 0;

int z = x / y;

Console.WriteLine("Result: " + z); }

catch (DivideByZeroException) {

Console.Write("Division by zero not allowed |");

} finally {

Console.Write(" Finally block executed"); } }}

Ans: Division by zero not allowed | Finally block executed

1. . Which exception occurs when you access an array element beyond its limit?

Ans: IndexOutOfRangeException

11. What does the keyword throw; inside a catch block do?

Ans: Rethrows the same exception

12. Predict the output try { int[] arr = { 10, 20, 30 }; Console.WriteLine(arr[3]); } catch (DivideByZeroException){ Console.WriteLine("Divide by zero"); } catch (IndexOutOfRangeException){ Console.WriteLine("Index error"); } finally{ Console.WriteLine("End of program"); }

Ans: Index error

End of program

13. What is the use of ApplicationException class?

Ans: Used for user-defined exceptions

14. 4. Predict the output try{ int x = int.Parse("123A"); Console.WriteLine("Number: " + x); } catch (FormatException){ Console.WriteLine("Invalid number format"); }

Ans: Invalid number format

15. Which block executes when an exception occurs in the try block?

Ans: Catch

16. In C#, every user-defined (custom) exception class must directly inherit from the System.Exception class or one of its derived classes.

Ans: True

17. What is exception propagation?

Ans: Passing an exception up the call stack until caught

18. Which block is optional in try-catch-finally structure?

Ans: Catch

19. What will happen if both try and finally blocks have return statements?

Ans: finally’s return overrides try’s

20. Which of the following statements about custom exceptions is correct?

Ans: Must inherit from Exception or ApplicationException