**Assignment-5**

**Q1. Handling Division by Zero Read 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.**

using System;

namespace Assignment\_5

{

internal class Ques1

{

static void Main(string[] args)

{

Console.Write("Enter numerator: ");

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

Console.Write("Enter denominator: ");

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

try

{

double result = a / b;

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

}

catch (DivideByZeroException)

{

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

}

finally

{

Console.WriteLine("Execution completed.");

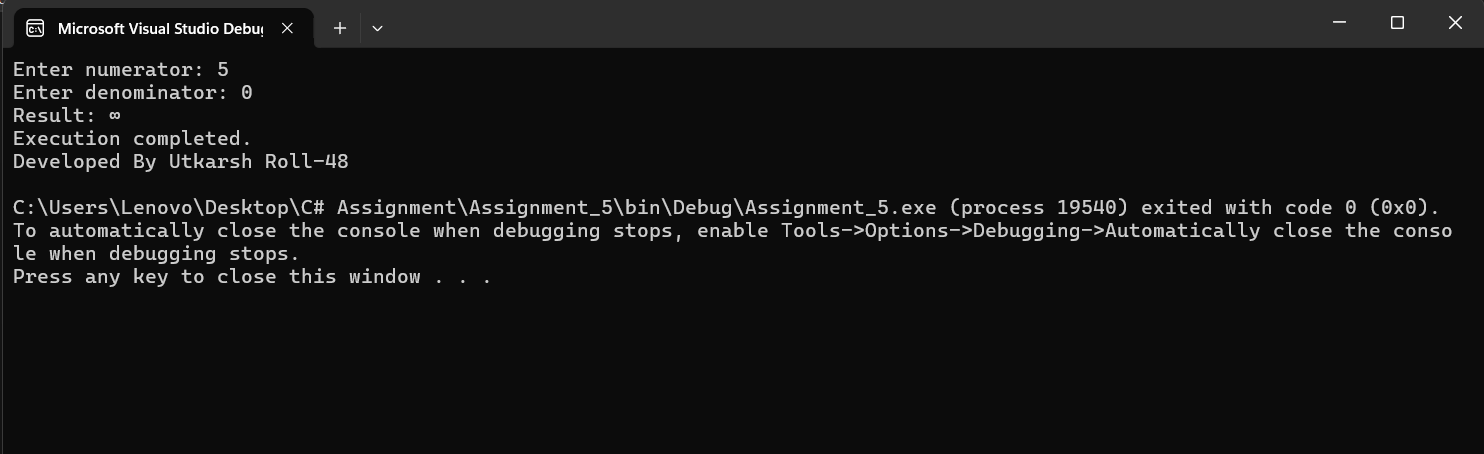
Console.WriteLine("Developed By Utkarsh Roll-48");

}

}

}

}



**Q2. Multiple Catch Blocks Read console input and convert to int. Handle FormatException, OverflowException, and a generic Exception, with distinct messages.**

using System;

namespace Assignment\_5

{

internal class Ques2

{

static void Main(string[] args)

{

Console.Write("Enter an integer: ");

string input = Console.ReadLine();

try

{

int value = int.Parse(input);

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

}

catch (FormatException)

{

Console.WriteLine("Invalid format: please enter digits only.");

}

catch (OverflowException)

{

Console.WriteLine("Overflow: number is too large or too small for Int32.");

}

catch (Exception ex)

{

Console.WriteLine("Unexpected error: " + ex.Message);

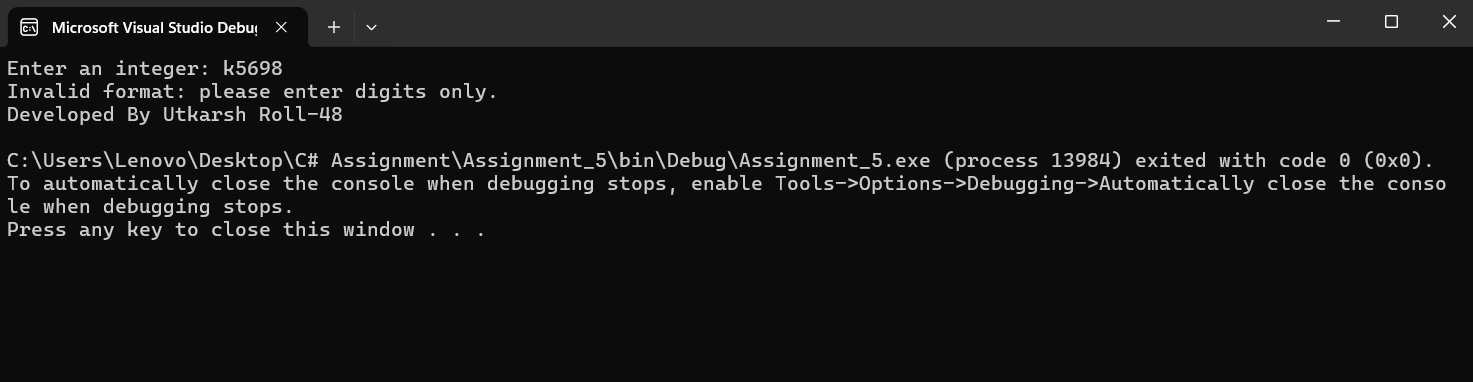
}

Console.WriteLine("Developed By Utkarsh Roll-48");

}

}

}



**Q3. Custom Exception — NegativeSalaryException Define NegativeSalaryException : Exception. If entered salary < 0, throw it and handle with a clear error message.**

using System;

namespace Assignment\_5

{

internal class Ques3

{

class NegativeSalaryException : Exception

{

public NegativeSalaryException() : base("Salary cannot be negative.") { }

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

}

static void Main(string[] args)

{

Console.Write("Enter salary: ");

string s = Console.ReadLine();

try

{

decimal salary = decimal.Parse(s);

if (salary < 0)

throw new NegativeSalaryException();

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

}

catch (NegativeSalaryException ex)

{

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

}

catch (FormatException)

{

Console.WriteLine("Invalid input: enter a numeric salary.");

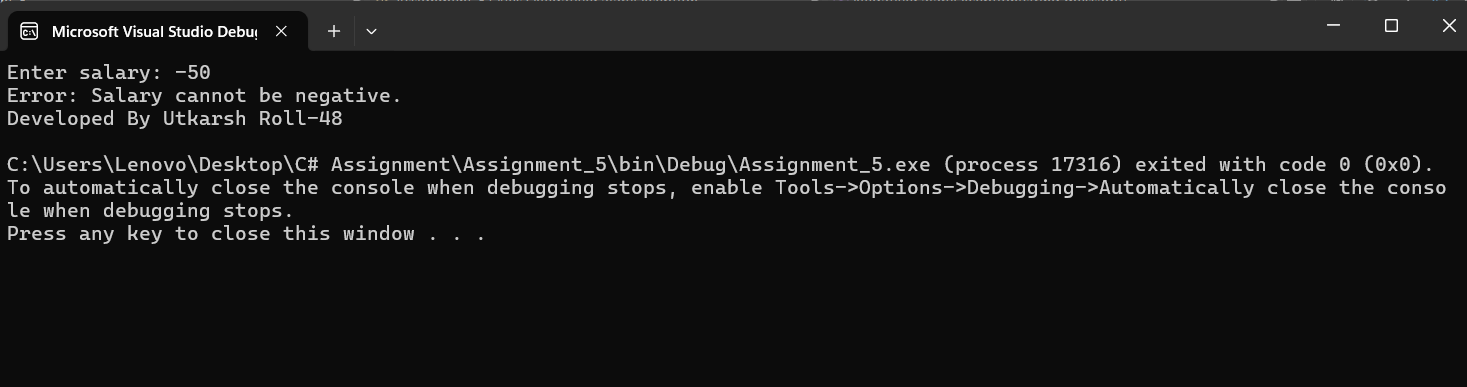
}

Console.WriteLine("Developed By Utkarsh Roll-48");

}

}

}



**Q4. Banking Scenario — InsufficientBalanceException Simulate withdrawal: if withdrawal > balance, throw custom InsufficientBalanceException; otherwise print remaining balance**.

using System;

namespace Assignment\_5

{

internal class Ques4

{

class InsufficientBalanceException : Exception

{

public InsufficientBalanceException() : base("Insufficient balance for this withdrawal.") { }

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

}

static void Main(string[] args)

{

Console.Write("Enter current balance: ");

decimal balance = decimal.Parse(Console.ReadLine());

Console.Write("Enter amount to withdraw: ");

decimal withdraw = decimal.Parse(Console.ReadLine());

try

{

if (withdraw > balance)

throw new InsufficientBalanceException();

balance -= withdraw;

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

}

catch (InsufficientBalanceException ex)

{

Console.WriteLine("Transaction failed: " + ex.Message);

}

catch (FormatException)

{

Console.WriteLine("Invalid input: use numbers for balance and withdrawal.");

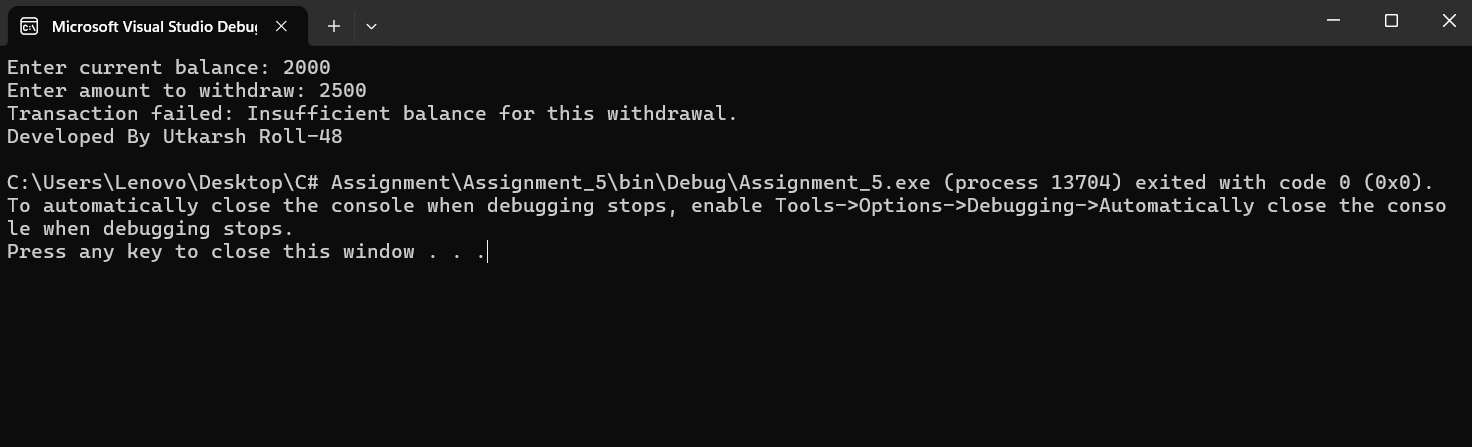
}

Console.WriteLine("Developed By Utkarsh Roll-48");

}

}

}



**Q5. Student Marks Validation Student class stores marks (0–100). If input outside range, throw InvalidMarksException. Demonstrate validation and handling in Main().**

using System;

namespace Assignment\_5

{

internal class Ques5

{

static void Main(string[] args)

{

try

{

Console.Write("Enter marks (0–100): ");

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

if (marks < 0 || marks > 100)

throw new Exception("Invalid Marks! Marks should be between 0 and 100.");

Console.WriteLine("Marks entered: " + marks);

}

catch (FormatException)

{

Console.WriteLine("Please enter numbers only.");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

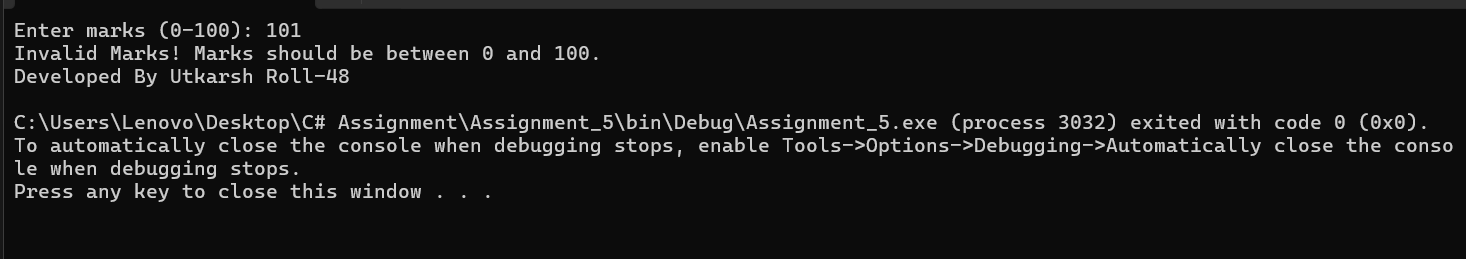
}

Console.WriteLine("Developed By Utkarsh Roll-48");

}

}

}

****

1. **C – catch**
2. **C – Executes always**
3. **B – Exception**
4. **A – Program terminates abnormally**
5. **B – throw**
6. **C – DivideByZeroException**
7. **B – More specific exceptions must appear before general ones**
8. **B – Yes**
9. **B – “Division by zero not allowed | Finally block executed”**
10. **A – IndexOutOfRangeException**
11. **A – Rethrows the same exception**
12. **B – “Index error” then “End of program”**
13. **B – User-defined exceptions**
14. **B – Invalid number format**
15. **C – catch**
16. **True**
17. **B – Passing up the call stack until caught**
18. **D – Both B and C**
19. **B – finally’s return overrides try’s**
20. **A – Must inherit from Exception or ApplicationException**