Lab 9: Module Dependency and Cohesion Analysis

Pathan Mohammad Rashid (22110187)

Introduction, Setup, and Tools

In this lab, I analyzed software dependencies and class cohesion using **pydeps** for a Python project and **LCOM** metrics for a Java project. My goals were to generate and interpret module dependency graphs, compute fan-in/fan-out, detect design flaws, and measure class cohesion to suggest refactoring strategies.

I selected the **Requests** library (a simple, yet elegant HTTP library) for Python dependency analysis, and the **Google Guava** library (core Java utilities) for cohesion analysis.

Environment and Tools:

- Operating System: SET-IITGN-VM (in Oracle VM VirtualBox)
- Python Version: 3.9.7
- Java Version: 11.0.16
- **pydeps** v1.3.6 (Python module dependency visualization)
- **LCOM.jar** v1.0 (LCOM1..5 & YALCOM analysis tool)

Setup & Installation:

```
# Update package lists
sudo apt update

# Install Python 3.8+, pip, and Graphviz
sudo apt install -y python3 python3-pip graphviz          # Graphviz
provides `dot` on PATH

# Install Java 11 (or later) and Maven
sudo apt install -y openjdk-11-jdk maven

pip3 install pydeps                          # installs the pydeps CLI
```

Methodology and Execution

1. Python Dependency Analysis with pydeps

Project Setup

I cloned the Requests repository from GitHub:

```
git clone https://github.com/psf/requests.git
cd requests
```

Listing 1: Cloning Requests Repository

Generating Dependency Data

I ran pydeps with the --show-deps option to output JSON for computing fan-in and fan-out:

```
pydeps . --show-deps --deps-output deps.json
```

Listing 2: Running pydeps with JSON output

Dependency Metrics

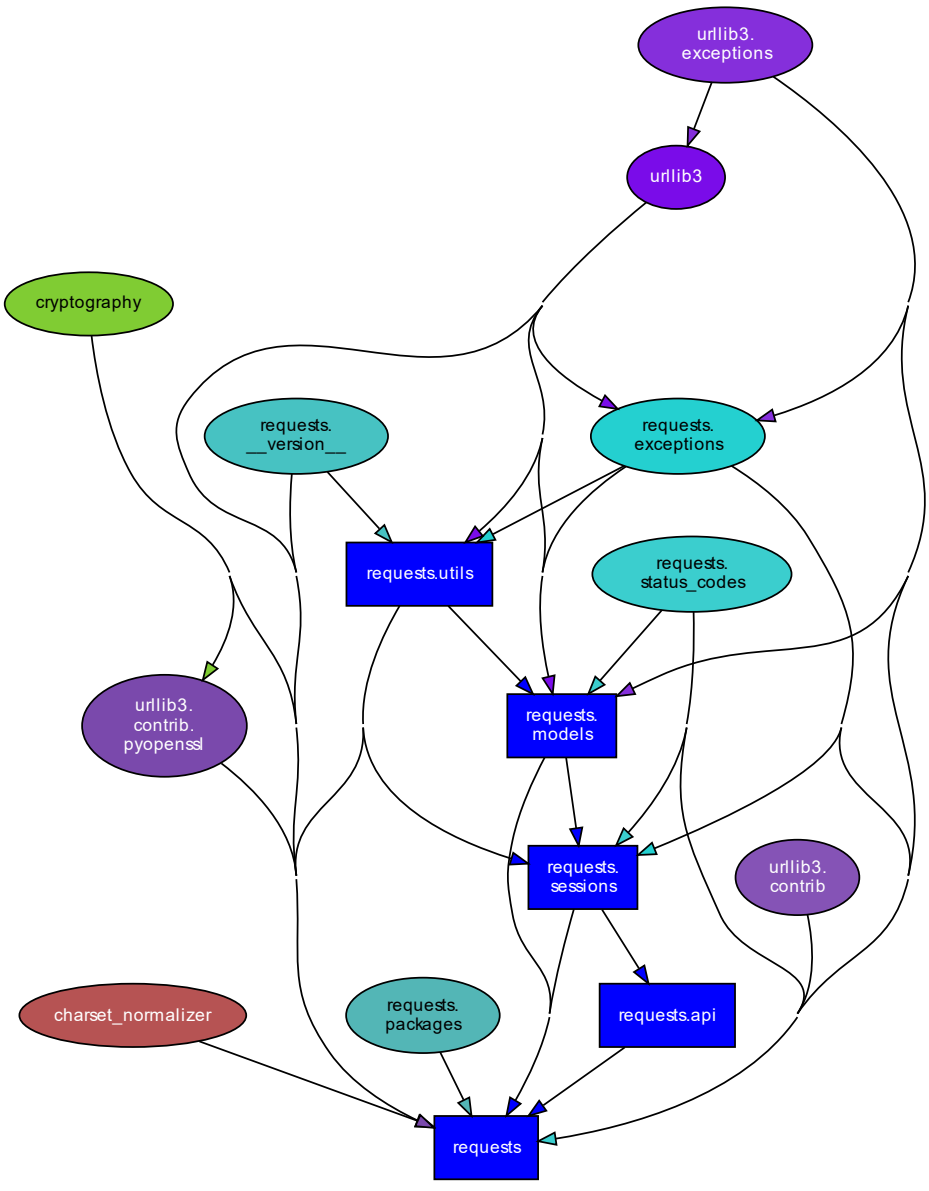


Table 1: Module Fan-In and Fan-Out (pydeps output)

Module	Fan-In	Fan-Out
main	0	1
charset_normalizer	1	0
cryptography	3	1
requests	4	15
requests._version_	2	0
requests.api	1	2
requests.exceptions	4	2
requests.models	2	5
requests.packages	1	0
requests.sessions	2	4
requests.status_codes	3	0
requests.utils	3	4
urllib3	6	2
urllib3.contrib	1	0
urllib3.contrib.pyopenssl	1	2
urllib3.exceptions	4	0

Analysis:

- **Highly Coupled Modules:** requests (Fan-In=4, Fan-Out=15) and urllib3 (Fan-In=6, Fan-Out=2) are central nodes in the dependency graph. requests is the most interconnected, highlighting its role as the core orchestrator.
- **Cyclic Dependencies:** None detected. The graph is acyclic, showing that the system avoids mutual dependencies that can complicate maintenance and testing.
- **Unused / Disconnected Modules:** Modules with Fan-Out = 0 (no dependencies): charset normalizer, requests._version_, requests.packages, requests.status codes, urllib3.contrib, urllib3.exceptions. These may be utility classes or legacy modules.
- **Dependency Depth:** The dependency graph exhibits a maximum depth of 2–3 levels. For example, cryptography and charset normalizer indirectly support modules like requests.utils, which in turn affect requests.models. The relatively shallow hierarchy favors ease of navigation and modular testing.

Impact Assessment

- **Core Module Risk:** The requests module, having the highest Fan-Out (15), acts as the core of the project. Any change in this module may propagate to requests.models, requests.sessions, requests.utils, requests.exceptions, and more, potentially causing widespread breakage or unexpected behavior.

- **Risky Dependencies:** urllib3 (Fan-In=6) is a critical low-level dependency. If modified without backward compatibility, it could break higher-level functionalities in multiple modules relying on its networking stack.

- **Module Isolation:** Modules with low Fan-In and Fan-Out (e.g., requests.status codes, urllib3.contrib) are relatively isolated, posing less risk in change scenarios.

2. Java Class Cohesion Analysis with LCOM

Project Setup

I cloned the Guava repository from GitHub:

```
git clone  https://github.com/google/guava.git
cd guava

git clone https://github.com/tushartushar/LCOM.git
cd LCOM
mvn clean package                    # produces target/LCOM.jar
```

Listing 3: Cloning Guava Repository

Running LCOM Analysis

I executed:

```
java -Xmx2g -jar /home/set-iitgn-vm/Desktop/Lab9/LCOM/target/LCOM.jar -i /home/set-iitgn-
    vm/Desktop/Lab9/LCOM/guava/guava/src -o lcom-output
```

Listing 4: Executing LCOM.jar on Guava

Cohesion Metrics

Java Class	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
`MutableClassToInstanceMap`	214.0	175.0	13.0	13.0	0.86	0.57
`SerializedForm`	0.0	0.0	1.0	1.0	1.0	0.0
`Range`	666.0	429.0	21.0	9.0	0.86	0.21
`RangeLexOrdering`	0.0	0.0	1.0	1.0	0.0	0.0
`AbstractRangeSet`	105.0	0.0	15.0	13.0	0.0	0.87
`FilteredKeySetMultimap`	36.0	0.0	9.0	9.0	0.0	0.67
`EntrySet`	1.0	0.0	2.0	2.0	0.0	1.0
`UnmodifiableListIterator`	3.0	0.0	3.0	3.0	0.0	1.0
`HashMultimapGwtSerializationDependencies`	0.0	0.0	1.0	1.0	0.0	0.0

Table 2: LCOM Metrics for Selected Guava Classes

Analysis: High LCOM values indicate low cohesion—classes like Range (LCOM1=666, YALCOM=0.209) and MutableClassToInstanceMap (LCOM1=214, YALCOM=0.565) are candidates for functional decomposition.

Results and Analysis

- **Dependency Analysis:** requests is highly coupled; no import cycles; several disconnected modules.
- **Cohesion Analysis:** Multiple Guava classes exhibit low cohesion (high LCOM), suggesting refactoring opportunities.

Discussion and Conclusion

This lab taught me the importance of managing coupling and cohesion to improve maintainability. Key challenges included parsing pydeps JSON output and interpreting large LCOM reports. I learned to prioritize low coupling to minimize change impact and to seek high cohesion by grouping related functionality. In future work, I will explore additional metrics such as cyclomatic complexity and automate refactoring suggestions.

References and Resources

- **Requests GitHub:** https://github.com/psf/requests
- **pydeps GitHub:** https://github.com/thebjorn/pydeps
- **Guava GitHub:** https://github.com/google/guava
- **LCOM GitHub:** https://github.com/tushartushar/LCOM.git

GitHub Repository and Drive Link

GitHub Repository: https://github.com/Pathan-Mohammad-Rashid/STT-Lab

Drive Link: 22110187_STT_Labs

Lab 10: Development of C# Console Applications

Pathan Mohammad Rashid (22110187)

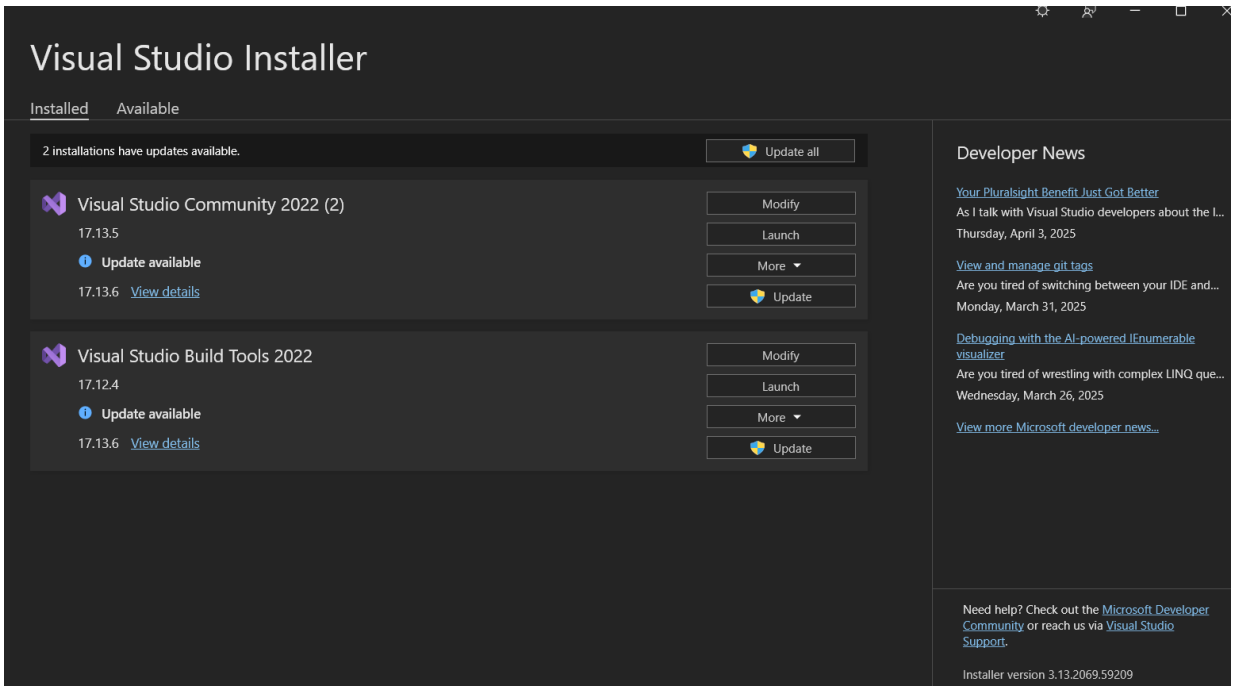
Introduction, Setup, and Tools

In this lab, I explore .NET development using C# in Visual Studio, focusing on creating console applications to grasp basic syntax, control structures, functions, object-oriented principles, exception handling, and debugging techniques. The objectives are to:

- Set up Visual Studio 2022 Community Edition for .NET development.
- Write and run C# console applications implementing arithmetic operations, loops, functions, and classes.
- Handle exceptions and use the Visual Studio Debugger (step-in, step-over, step-out).

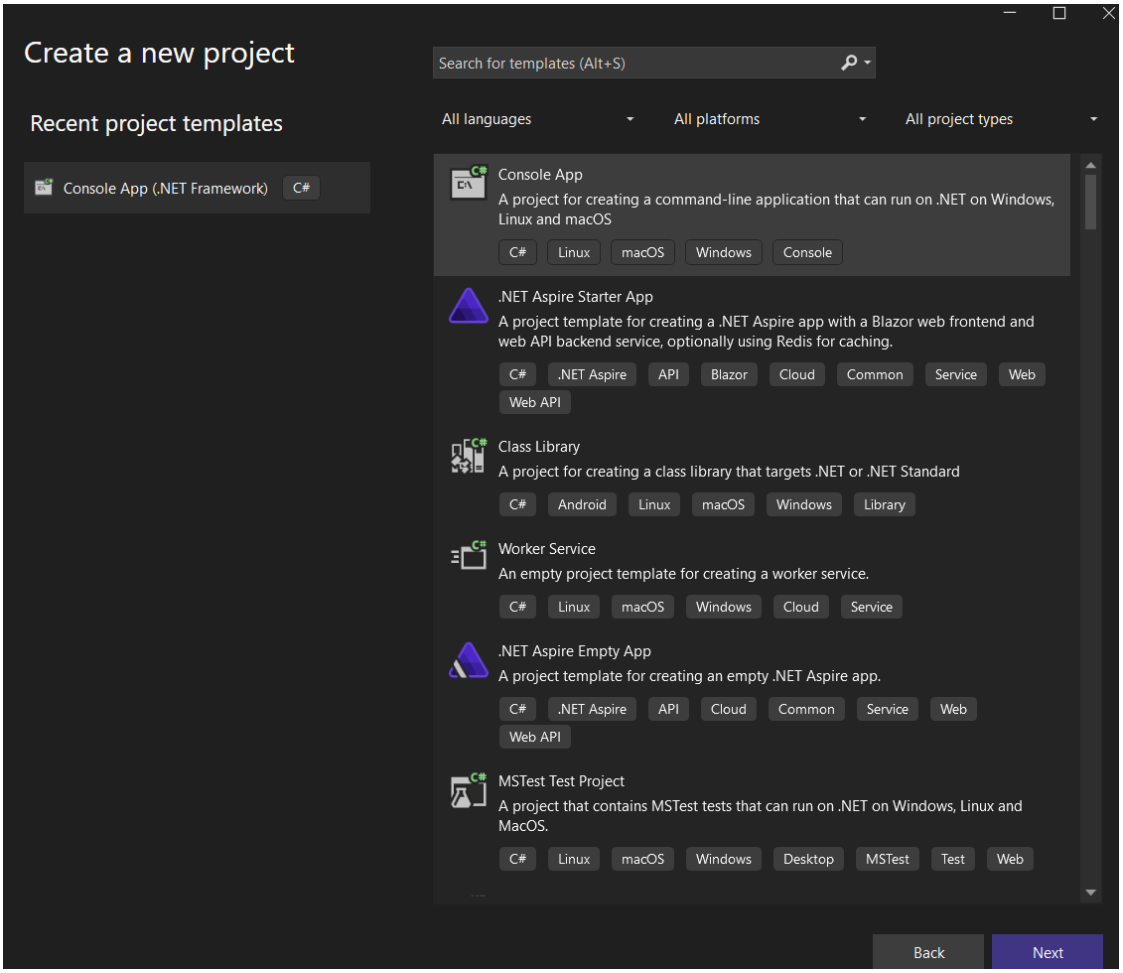
The environment and tools used were:

- **Operating System:** Windows 11.
- **IDE:** Visual Studio 2022 Community Edition.
- **Framework:** .NET 6.0 SDK (v6.0.410) with C# 10.0 support.
- **Language Version:** C# 13.0 (latest stable as of January 2025).

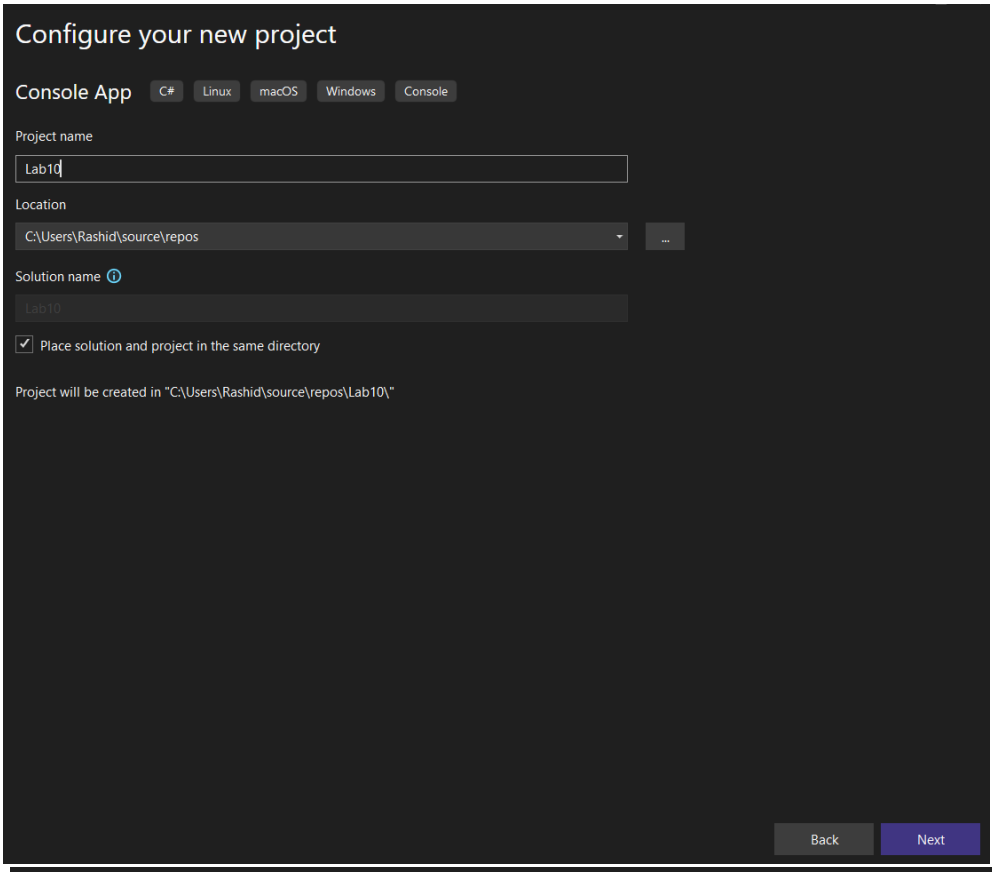


Methodology and Execution

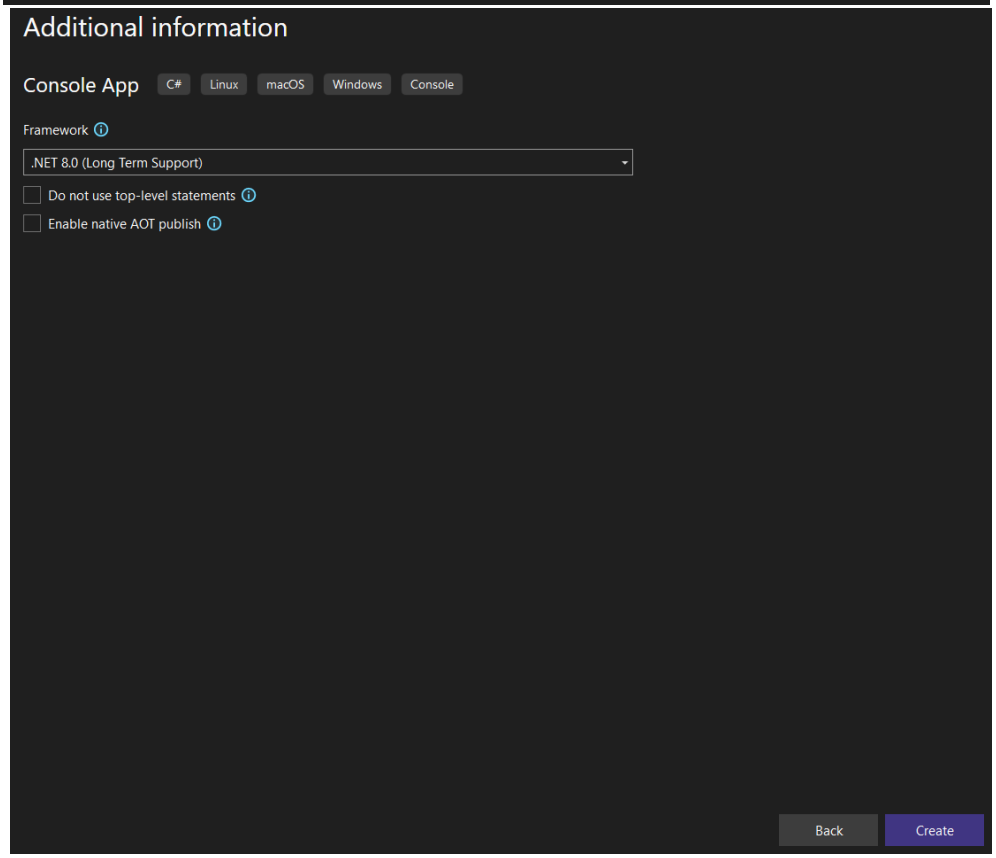
Activity 1: Setting Up .NET Development Environment

1. Open Visual Studio 2022, select *Create a new project → Console App (.NET)*.
2. Choose *.NET 6.0* as the target framework.
3. Name the project Lab10 and create it.
4. 

5.


Configure your new project
Console App
C#
Linux
macOS
Windows
Console
Project name
Lab10
Location
C:\Users\Rashid\source\repos
...
Solution name
Lab10
Place solution and project in the same directory
Project will be created in "C:\Users\Rashid\source\repos\Lab10\"
Back
Next

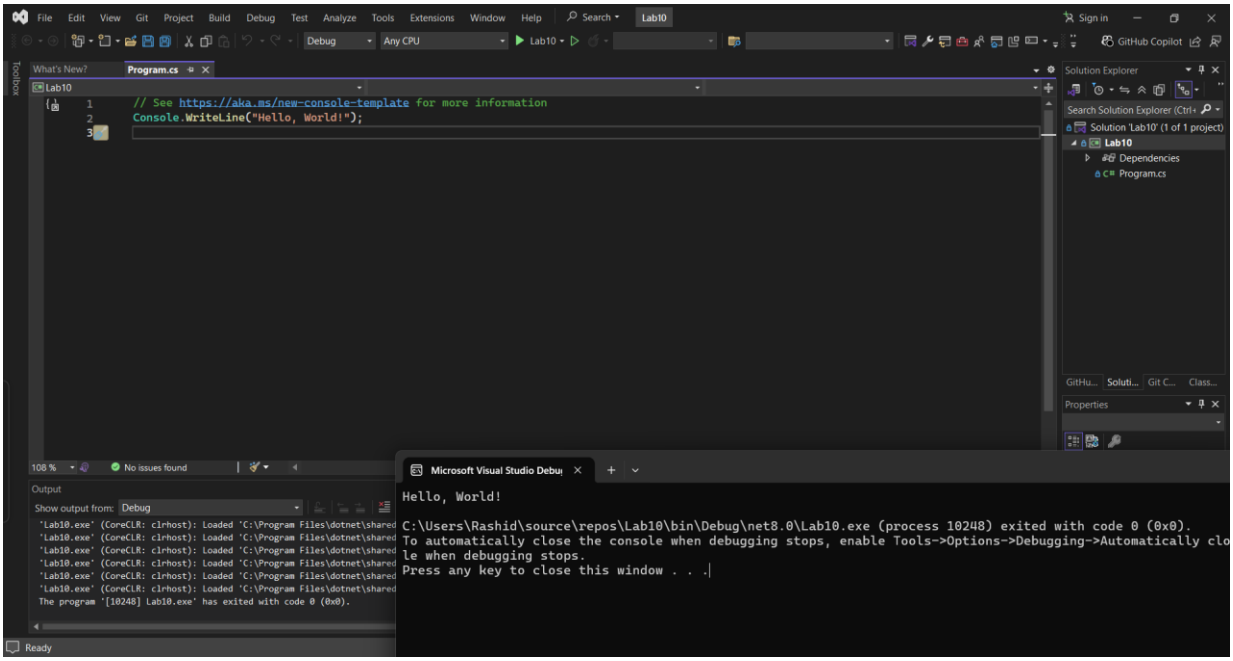
6.

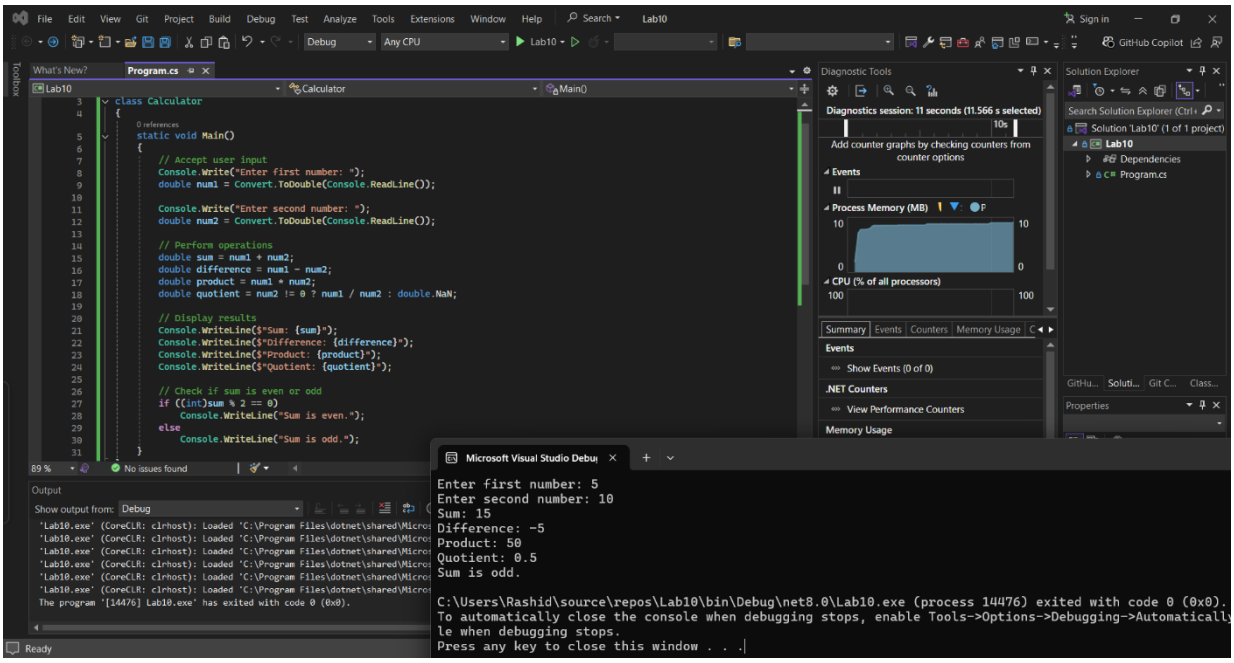

Additional information
Console App
C#
Linux
macOS
Windows
Console
Framework
.NET 8.0 (Long Term Support)
Do not use top-level statements
Enable native AOT publish
Back
Create

Activity 2: Basic Syntax and Control Structures

Requirements: Accept two numbers, perform addition, subtraction, multiplication, division; determine if the sum is even or odd; display all results. **Workflow:**

- Prompt user for input using Console.ReadLine().
- Parse inputs to int.
- Compute operations and use if-else to check sum parity.
- Output via Console.WriteLine().





2

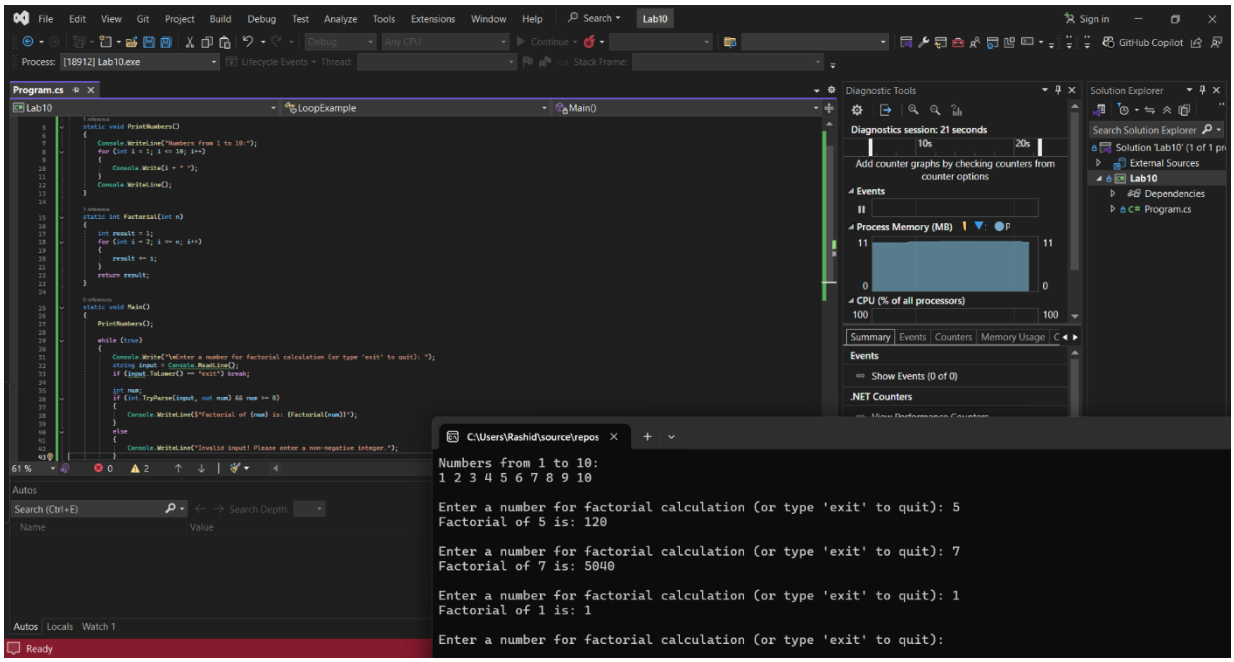
Listing 1: ArithmeticOperations.cs

Activity 3: Loops and Functions

Requirements: Print numbers 1–10 with a for loop; repeatedly prompt until user enters "exit" using a while loop; define and call a Factorial function. **Workflow:**

- Use for (int i = 1; i <= 10; i++) to print numbers.
- Implement while(true) loop, break on "exit".
- Define static long Factorial(int n) using recursion or iteration.

Code Snippet:

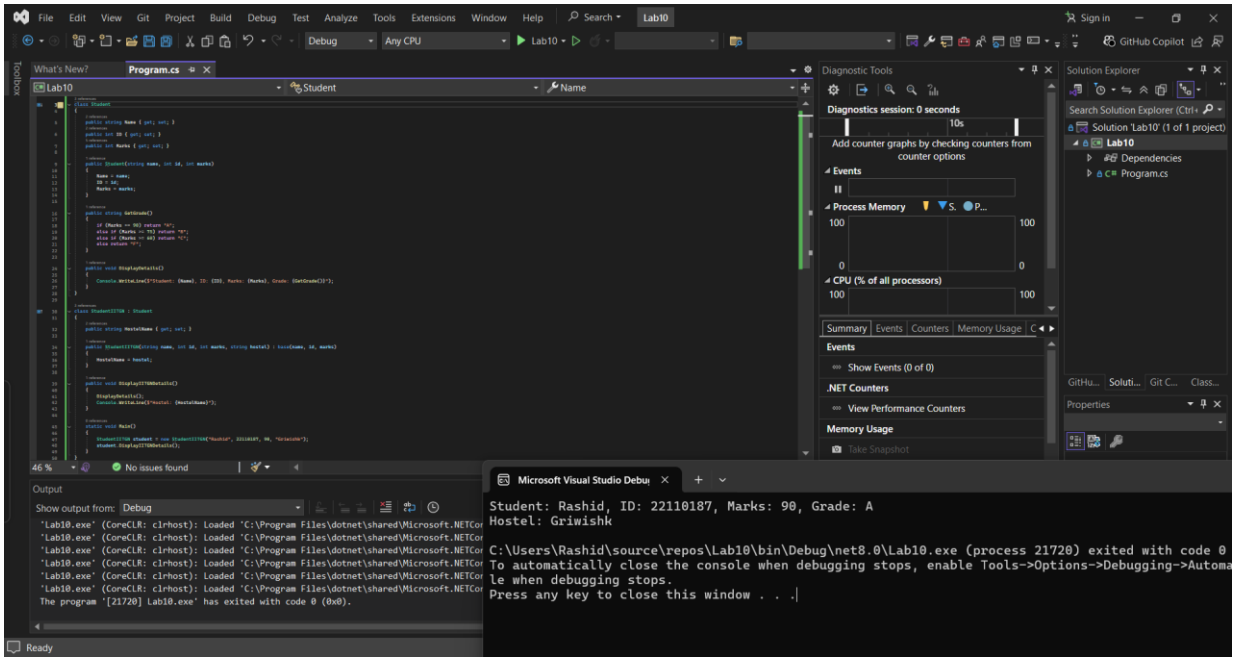


Listing 2: LoopsAndFunctions.cs

Activity 4: Object-Oriented Programming

Requirements: Create Student class with Name, ID, Marks, constructor, GetGrade(), and Main(); derive StudentIITGN adding Hostel Name IITGN, with its own Main(). **Workflow:**

- Define public class Student with properties and constructor.
- Overload constructors including a copy constructor.
- Implement GetGrade() returning A/B/C based on marks.
- Define StudentIITGN : Student adding hostel property.
- Observe behavior when both classes have static void Main().

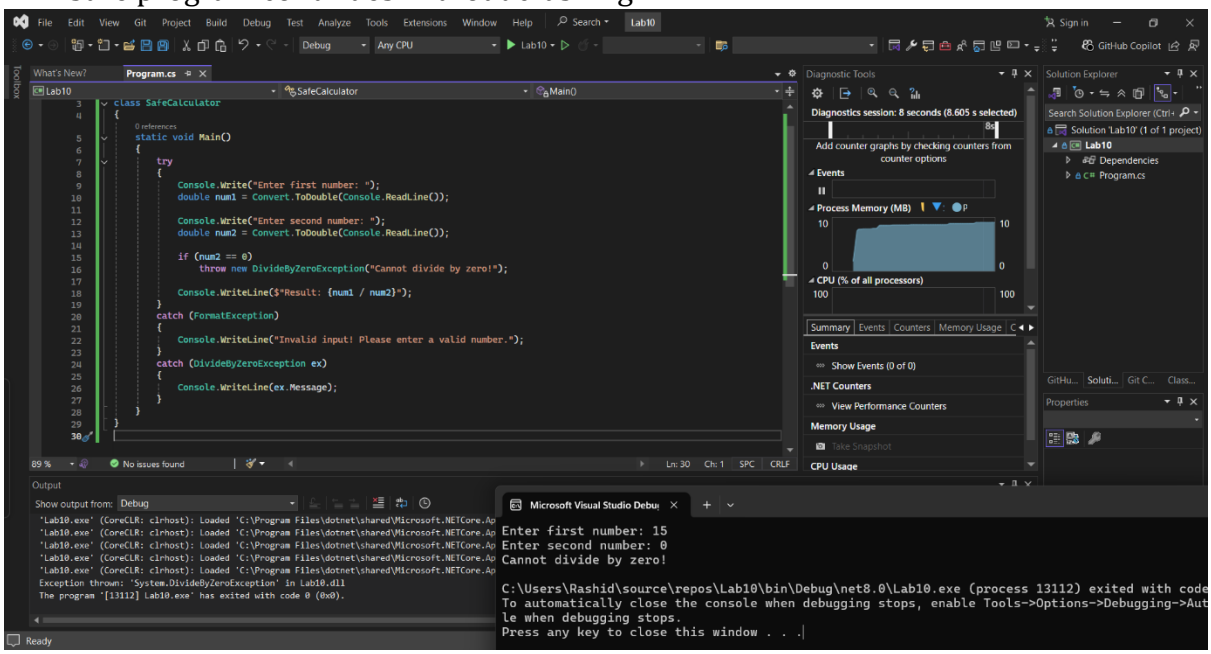


Listing 3: StudentClasses.cs

Activity 5: Exception Handling

Requirements: Modify Activity 2 program to handle division-by-zero and invalid input using try-catch. **Workflow:**

- Wrap parsing and division in try block.
- Catch FormatException for invalid numbers.
- Catch DivideByZeroException for zero divisor.
- Ensure program continues without crashing.

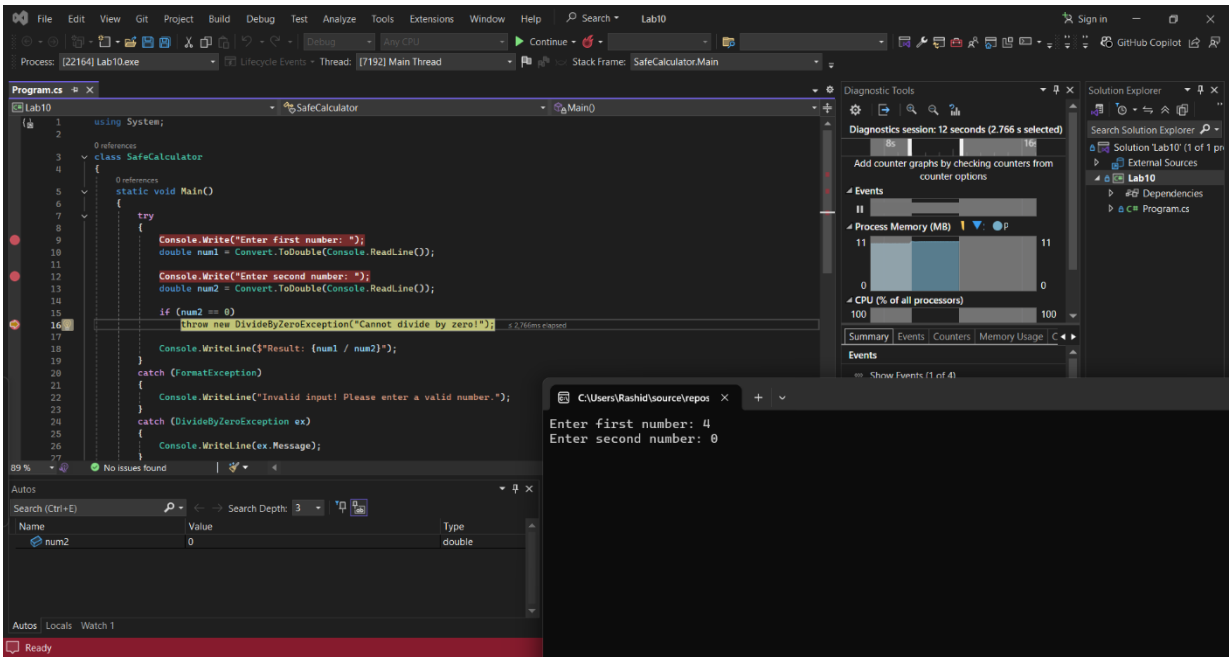


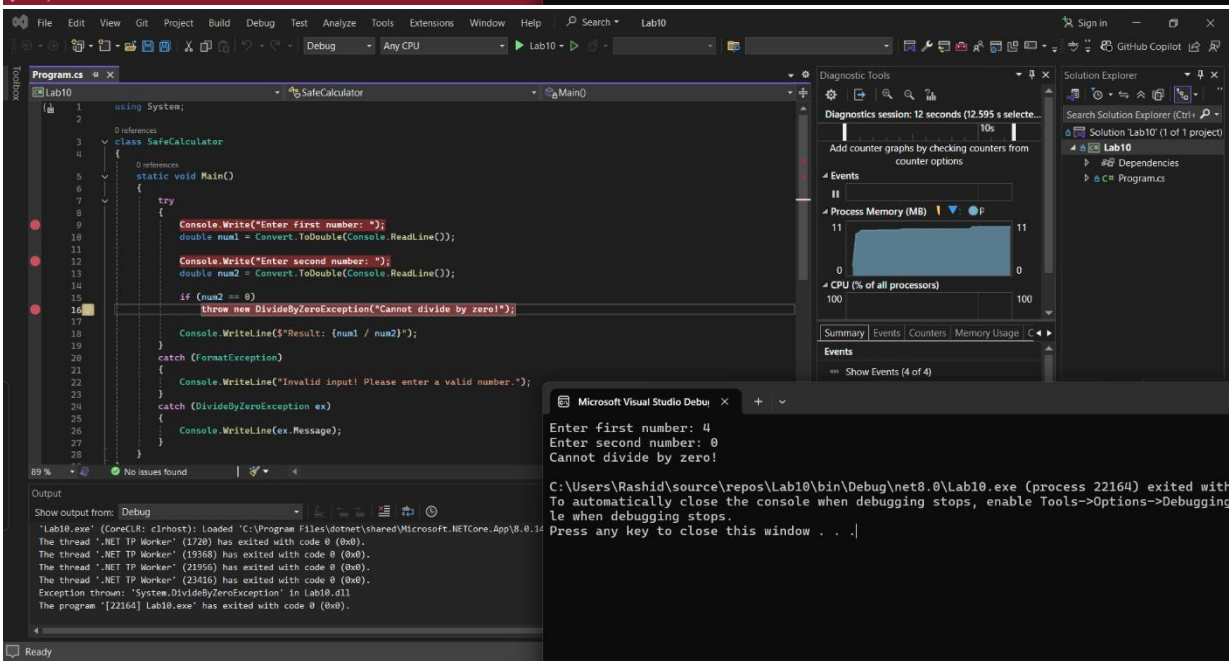
Listing 4: ExceptionHandling.cs

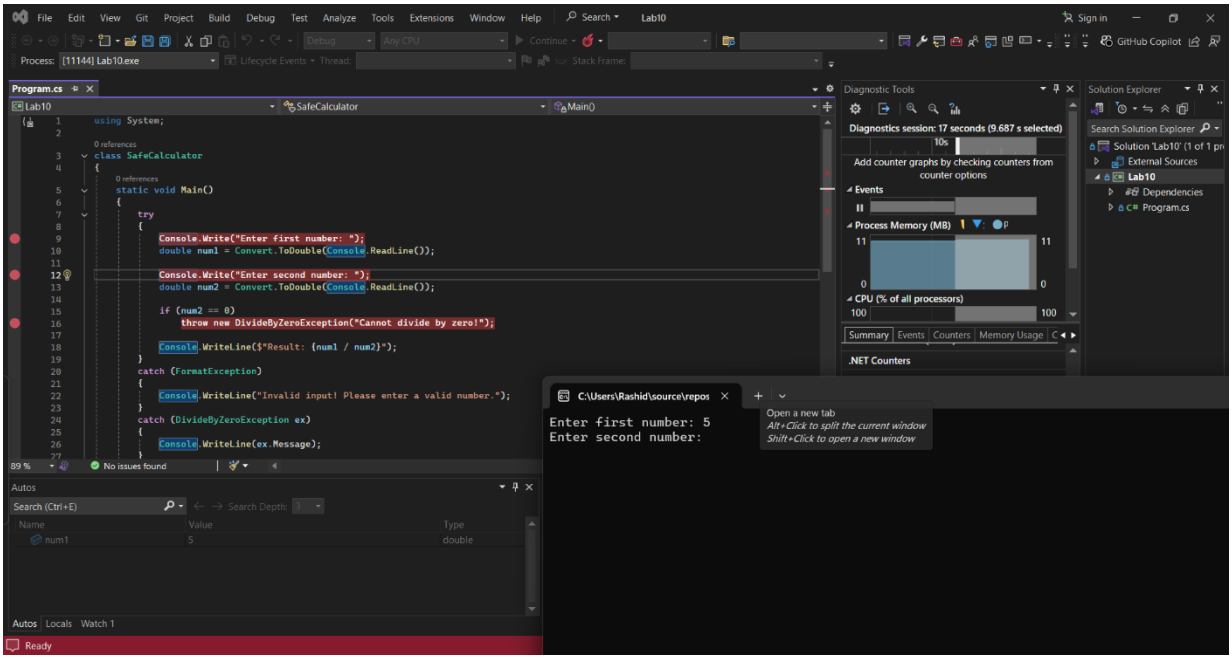
Activity 6: Debugging using Visual Studio Debugger

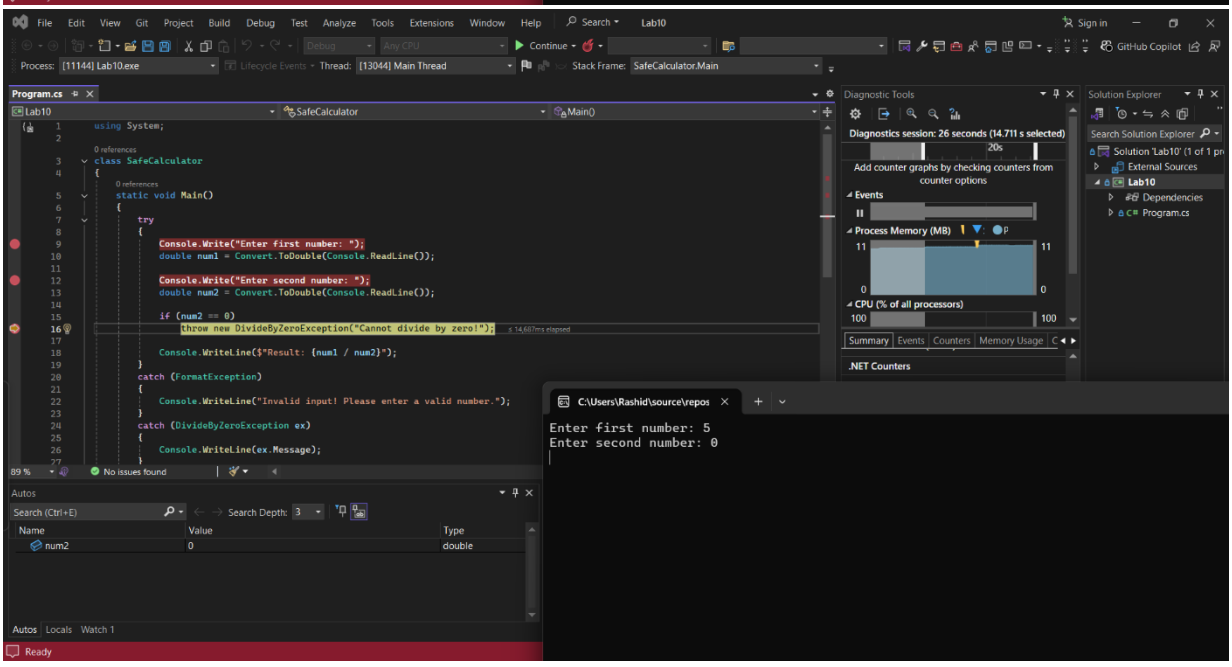
Requirements: Insert breakpoints in Activities 2–5, demonstrate step-in, step-over, step-out.

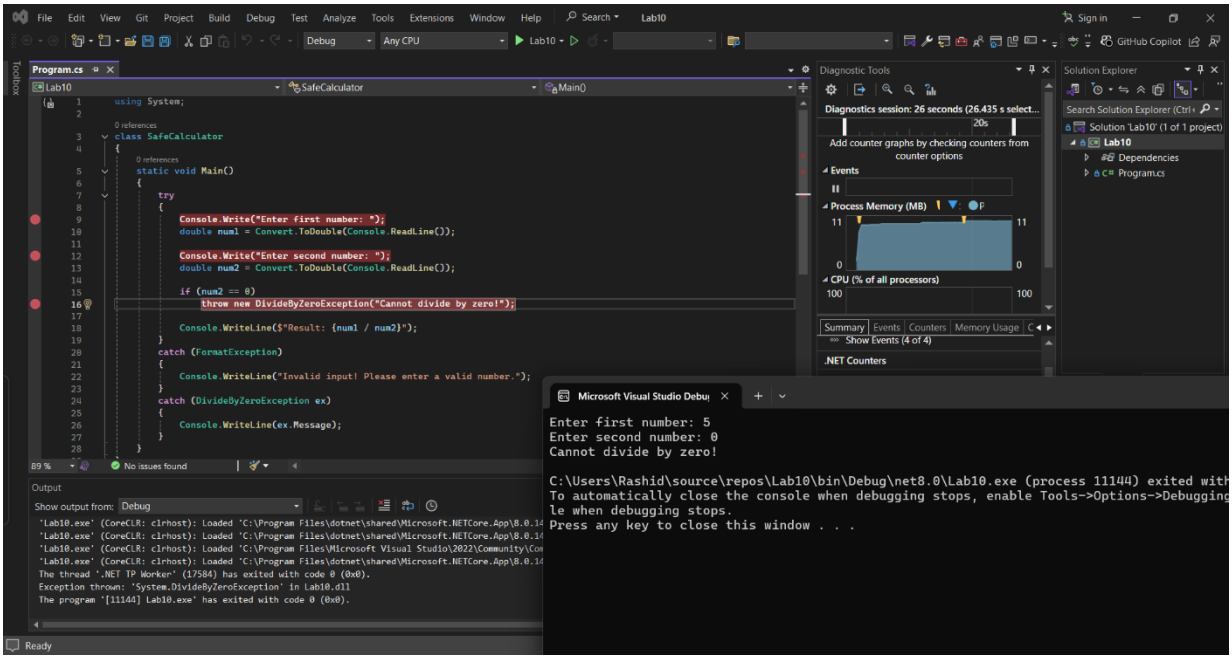
Workflow:






Program.cs
Lab10
SafeCalculator
Main()
using System;
class SafeCalculator
static void Main()
try
Console.Write("Enter first number: ");
double num1 = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter second number: ");
double num2 = Convert.ToDouble(Console.ReadLine());
if (num2 == 0)
throw new DivideByZeroException("Cannot divide by zero!");
Console.WriteLine($"Result: {num1 / num2}");
catch (FormatException)
Console.WriteLine("Invalid input! Please enter a valid number.");
catch (DivideByZeroException ex)
Console.WriteLine(ex.Message);
Diagnostic Tools
Diagnostics session: 17 seconds (9.687 s selected)
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
.NET Counters
Solution Explorer
Solution 'Lab10' (1 of 1 pr
External Sources
Lab10
Dependencies
Program.cs
C:\Users\Rashid\source\repos
Enter first number: 5
Enter second number:
Open a new tab
Alt+Click to split the current window
Shift+Click to open a new window
Autos
Search (Ctrl+E)
Name
Value
Type
num1
5
double
Autos
Locals
Watch 1
Ready


Program.cs
Lab10
SafeCalculator
Main()
using System;
class SafeCalculator
static void Main()
try
Console.Write("Enter first number: ");
double num1 = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter second number: ");
double num2 = Convert.ToDouble(Console.ReadLine());
if (num2 == 0)
throw new DivideByZeroException("Cannot divide by zero!");
Console.WriteLine($"Result: {num1 / num2}");
catch (FormatException)
Console.WriteLine("Invalid input! Please enter a valid number.");
catch (DivideByZeroException ex)
Console.WriteLine(ex.Message);
Diagnostic Tools
Diagnostics session: 26 seconds (14.711 s selected)
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
.NET Counters
Solution Explorer
Solution 'Lab10' (1 of 1 pr
External Sources
Lab10
Dependencies
Program.cs
C:\Users\Rashid\source\repos
Enter first number: 5
Enter second number: 0
Autos
Search (Ctrl+E)
Name
Value
Type
num2
0
double
Autos
Locals
Watch 1
Ready



- Set breakpoints on input parsing, operations, function calls.
- Use *Step Into* to enter methods, *Step Over* to execute lines, *Step Out* to exit methods.

Results and Analysis

- **Arithmetic Operations:** Correct results for addition, subtraction, multiplication, division; sum parity accurately determined.
- **Loops & Functions:** Numbers 1–10 printed; prompt loop exited on "exit"; factorial function produced expected values (e.g., 5! = 120).
- **OOP:** Student and StudentIITGN instances initialized correctly; GetGrade() returned proper letter grades.
- **Exception Handling:** Program caught invalid input and division-by-zero without terminating.
- **Debugging:** Breakpoints and stepping functions worked as intended, facilitating inspection of variables and call flow.

Discussion and Conclusion

This lab deepened my understanding of C# console applications, object-oriented design, exception safety, and debugging. Key challenges included handling invalid user input gracefully and configuring breakpoints effectively. Lessons learned:

- Importance of input validation and exception handling for robust applications.
- Value of the Visual Studio Debugger in tracing execution and inspecting state.

- Utility of constructors (including copy constructors) and inheritance for code reuse.

In conclusion, mastering these fundamentals prepares for more advanced .NET development tasks and reinforces best practices in software engineering.

References and Resources

- Microsoft Learn: C# Fundamentals.
- Microsoft Learn: Exception Handling in C#.
- Microsoft Learn: Visual Studio Debugger.
- Microsoft Learn: C# Inheritance Tutorial.

GitHub Repository and Drive Link

GitHub Repository: https://github.com/Pathan-Mohammad-Rashid/STT-Lab

Drive Link: 22110187_STT_Labs