CS202: Software Tools and Techniques for CSE

Lab Assignment 9

By Sriram Srinivasan (22110258)

Introduction, Setup, and Tools

Objective

This laboratory experiment aims to analyze software architecture quality through dependency mapping and cohesion measurement. The primary goals are to identify structural issues like cyclic dependencies, locate highly coupled modules, assess class cohesion, and recommend potential refactoring strategies to improve software design.

Overview

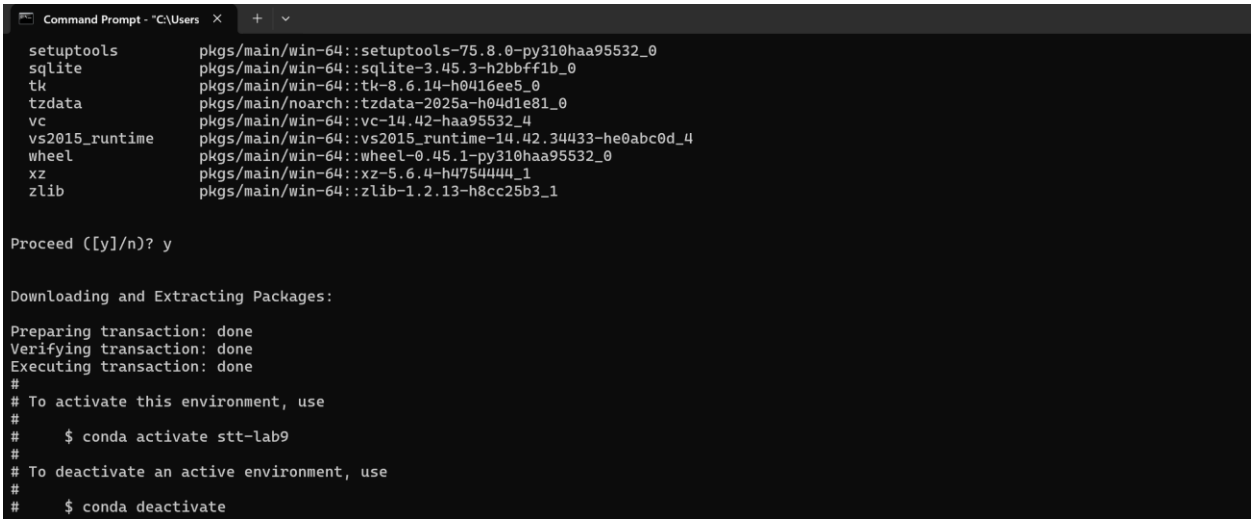
This lab examines two distinct open-source projects: Flask-SQLAlchemy (Python) for dependency analysis and Spring PetClinic (Java) for cohesion evaluation. By applying specialized tools to analyze these codebases, we gain insights into their architectural health and identify opportunities for structural improvement. The analysis focuses particularly on module coupling, dependency cycles, and class cohesion - key indicators of software maintainability and extensibility.

Tools and Versions

- **Operating System**: Windows 11
- **Python Environment**: Python 3.10.2
- **Dependency Analysis Tool**: pydeps
- **Cohesion Analysis Tool**: LCOM.jar

Methodology and Execution

Conda Installation




```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9>git clone https://github.com/pallets-eco/flask-sqlalchemy
Cloning into 'flask-sqlalchemy'...
remote: Enumerating objects: 4969, done.
remote: Counting objects: 100% (719/719), done.
remote: Compressing objects: 100% (91/91), done.
remote: Total 4969 (delta 654), reused 628 (delta 628), pack-reused 4250 (from 3)
Receiving objects: 100% (4969/4969), 1.63 MiB | 3.96 MiB/s, done.
Resolving deltas: 100% (2954/2954), done.
```

For this analysis, I selected "**Flask-SQLAlchemy**" - a popular extension for Flask that adds SQLAlchemy support. This project consists of multiple modules organized in folders and contains approximately 3,500+ lines of code, meeting the minimum requirement of 500+ lines specified in the assignment. (https://github.com/pallets-eco/flask-sqlalchemy)

Installation of PyDeps

```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9>pip install pydeps
Collecting pydeps
  Downloading pydeps-3.0.1-py3-none-any.whl.metadata (22 kB)
Collecting stdlib_list (from pydeps)
  Downloading stdlib_list-0.11.1-py3-none-any.whl.metadata (3.3 kB)
Downloading pydeps-3.0.1-py3-none-any.whl (47 kB)
Downloading stdlib_list-0.11.1-py3-none-any.whl (83 kB)
Installing collected packages: stdlib_list, pydeps
Successfully installed pydeps-3.0.1 stdlib_list-0.11.1
```

```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy>cd src

(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src>dir
 Volume in drive C is OS
 Volume Serial Number is CE9B-3985

 Directory of C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src

10-04-2025  21:36    <DIR>          .
10-04-2025  21:36    <DIR>          ..
10-04-2025  21:36    <DIR>          flask_sqlalchemy
               0 File(s)              0 bytes
               3 Dir(s)  82,521,341,952 bytes free

(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src>cd flask_sqlalchemy

(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src\flask_sqlalchemy>dir
 Volume in drive C is OS
 Volume Serial Number is CE9B-3985

 Directory of C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src\flask_sqlalchemy

10-04-2025  21:36    <DIR>          .
10-04-2025  21:36    <DIR>          ..
10-04-2025  21:36               500 cli.py
10-04-2025  21:36            39,446 extension.py
10-04-2025  21:36            11,733 model.py
10-04-2025  21:36            11,483 pagination.py
10-04-2025  21:36                 0 py.typed
10-04-2025  21:36             3,853 query.py
10-04-2025  21:36             3,637 record_queries.py
10-04-2025  21:36             3,537 session.py
10-04-2025  21:36               871 table.py
10-04-2025  21:36             2,843 track_modifications.py
10-04-2025  21:36               679 __init__.py
              11 File(s)         78,582 bytes
               2 Dir(s)  82,521,407,488 bytes free

(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src\flask_sqlalchemy>
```

Once the repository is cloned on the local system, we understand its file structure. On inspection, we realise that all the Python files are inside **.\src\flask_sqlalchemy.** We move into the desired directory and run the pydeps command to store the dependencies in JSON format and also in a SVG file.

```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src>pydeps flask_sqlalchemy --show-deps --no-output --deps-output
 alchemy_deps.json
```

```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src>pydeps flask_sqlalchemy --show-deps -o alchemy_deps2.svg
{
    "__main__": {
        "bacon": 0,
        "imports": [
            "flask_sqlalchemy",
            "flask_sqlalchemy.cli",
            "flask_sqlalchemy.extension",
            "flask_sqlalchemy.model",
            "flask_sqlalchemy.pagination",
            "flask_sqlalchemy.query",
            "flask_sqlalchemy.record_queries",
            "flask_sqlalchemy.session",
            "flask_sqlalchemy.table",
            "flask_sqlalchemy.track_modifications"
        ],
        "name": "__main__"
```

Fan-In and Fan-Out of the Modules

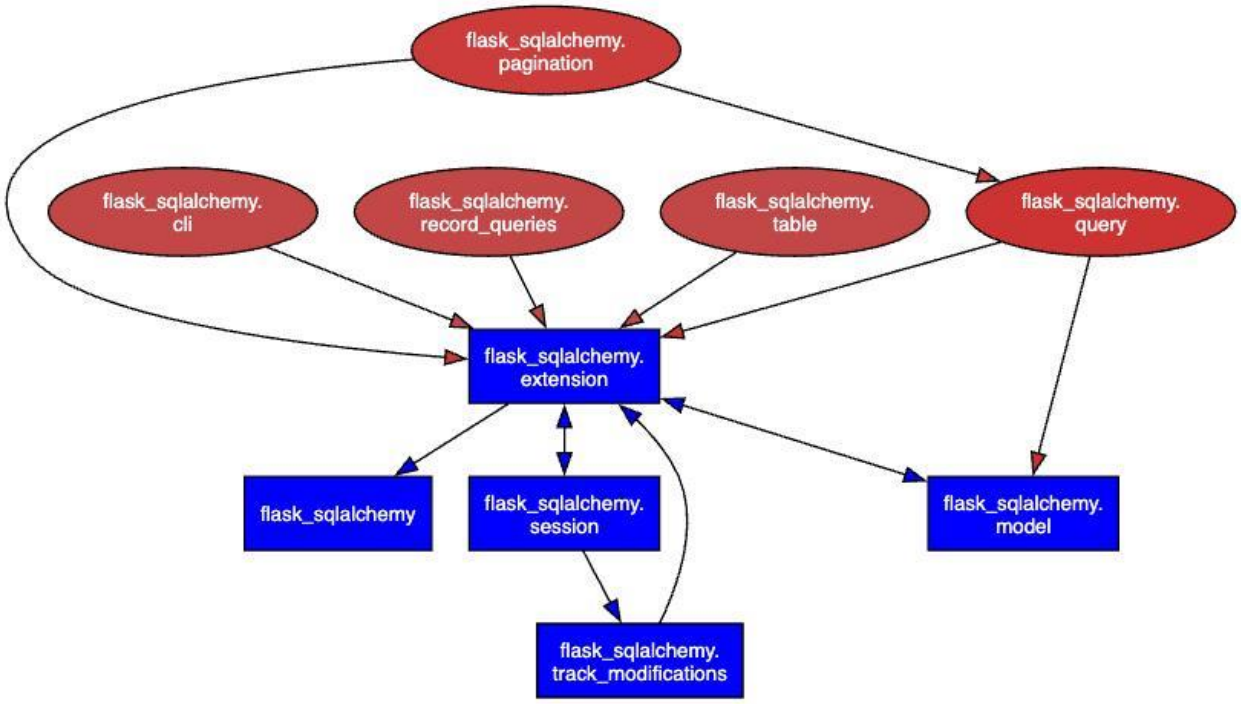
```
(stt-lab9) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\flask-sqlalchemy\src>python analysis.py

Module Dependency Analysis for Flask-SQLAlchemy:
============================================================
Module Name                              | Fan-In   | Fan-Out
------------------------------------------------------------
flask_sqlalchemy.extension               | 4        | 9
__main__                                 | 0        | 10
flask_sqlalchemy.model                   | 2        | 2
flask_sqlalchemy.query                   | 3        | 1
flask_sqlalchemy.session                 | 3        | 1
flask_sqlalchemy                         | 2        | 1
flask_sqlalchemy.pagination              | 3        | 0
flask_sqlalchemy.track_modifications     | 2        | 1
flask_sqlalchemy.cli                     | 2        | 0
flask_sqlalchemy.record_queries          | 2        | 0
flask_sqlalchemy.table                   | 2        | 0

Detailed metrics saved to flask_sqlalchemy_dependency_metrics.json

Core Modules Analysis:
------------------------------------------------------------
Core Module: flask_sqlalchemy.extension
  Fan-In: 4 | Fan-Out: 9
  Impact Level: High
----------------------------------------
Core Module: __main__
  Fan-In: 0 | Fan-Out: 10
  Impact Level: High
----------------------------------------
Core Module: flask_sqlalchemy.query
  Fan-In: 3 | Fan-Out: 1
  Impact Level: Medium
----------------------------------------
Core Module: flask_sqlalchemy.session
  Fan-In: 3 | Fan-Out: 1
  Impact Level: Medium
----------------------------------------
Core Module: flask_sqlalchemy.pagination
  Fan-In: 3 | Fan-Out: 0
  Impact Level: Medium
----------------------------------------
```

A Python script analysis.py was written which tabulates the fan-in and fan-outs of the modules using the created json file.



Module Dependency Analysis for Flask-SQLAlchemy

Identification of Highly Coupled Modules

Upon analyzing the dependency graph for the Flask-SQLAlchemy project, several modules demonstrate significant coupling within the system architecture. The coupling patterns reveal important insights about the structural design and potential areas for architectural improvement.

Based on the provided dependency data, the modules with the highest coupling are:

- **flask_sqlalchemy.extension**: This module exhibits the highest coupling with 9 imports and 4 modules importing it. The extension module serves as a central component of the system, facilitating interactions between numerous other modules.
- **flask_sqlalchemy.model**: This module demonstrates moderate coupling with 2 imports and 3 modules importing it. It maintains important relationships with the extension and query modules.
- **flask_sqlalchemy.session**: With 1 import and 3 modules importing it, this module plays a critical role in the system's session management functionality.
- **flask_sqlalchemy.query**: This module shows a balanced coupling pattern with 1 import and 3 modules importing it, serving as an important query handling component.

The coupling patterns suggest that flask_sqlalchemy.extension acts as a hub in the system architecture, with most other modules either importing it directly or being imported by it. This centralized design creates a star-like dependency structure with the extension module at its core.

Detection of Cyclic Dependencies

The dependency analysis reveals several concerning cyclic dependencies that could potentially impact system maintainability:

1. **flask_sqlalchemy ↔ flask_sqlalchemy.extension**: These modules import each other, creating a direct circular dependency.
2. **flask_sqlalchemy.extension ↔ flask_sqlalchemy.model**: These modules form another circular dependency path.

3. **flask_sqlalchemy.extension ↔ flask_sqlalchemy.session**: These modules also exhibit circular dependency.

These cyclic dependencies introduce several maintenance challenges:

- **Initialization Complexity**: Circular dependencies can cause initialization problems where modules depend on each other to be fully loaded before they themselves can be fully loaded.
- **Refactoring Difficulties**: Changes to one module in a cycle necessitate careful consideration of impacts on all other modules in that cycle, significantly increasing the complexity of refactoring efforts.
- **Testing Complications**: Modules involved in cyclic dependencies are difficult to test in isolation, often requiring complex mocking or test setup.
- **Conceptual Confusion**: Circular dependencies often indicate unclear separation of concerns, where responsibilities between modules are not cleanly defined.

These cyclic dependencies represent a significant maintenance risk and suggest opportunities for architectural improvement through more careful separation of concerns.

Analysis of Unused and Disconnected Modules

The dependency graph does not reveal any completely unused or disconnected modules. All modules appear to be integrated into the system's dependency structure, either through importing other modules or being imported by them. This level of connectedness suggests a cohesive design where all components serve an active purpose within the system.

However, it's worth noting that some modules have paths listed as "null" in the JSON data:

- **main**: Has null path, which is expected as it represents the entry point rather than a physical module.
- **flask_sqlalchemy.pagination**: Has null path, which might indicate an external dependency or a module that is referenced but not directly part of the project's file structure.

While these modules with null paths are not technically disconnected (they participate in the import relationships), their null path status warrants further investigation to ensure they represent intentional design choices rather than potential issues.

Assessment of Dependency Depth

The dependency graph exhibits a relatively shallow depth structure, which is beneficial for maintainability. The maximum dependency chain appears to be:

1. **main** imports **flask_sqlalchemy.model**
2. **flask_sqlalchemy.model** imports **flask_sqlalchemy.query**
3. **flask_sqlalchemy.query** imports **flask_sqlalchemy.pagination**

This represents a maximum depth of 3 levels, which is relatively manageable compared to deep dependency hierarchies that can emerge in larger systems.

However, the cyclic dependencies previously identified effectively create infinite depth potential in certain module relationships, complicating the depth assessment. These cycles make it difficult to establish a clear hierarchical structure for the affected modules, which can impede understanding of the system architecture.

Dependency Impact Assessment

Core Module Impact Analysis

The flask_sqlalchemy.extension module serves as the core module of the system, with the highest number of dependencies both inward and outward. Changes to this module would have far-reaching effects:

1. **Direct Impact**: Changes would immediately affect flask_sqlalchemy, flask_sqlalchemy.model, and flask_sqlalchemy.session, which directly import this module.
2. **Indirect Impact**: Due to the interconnected nature of the system, changes could propagate through these dependencies to affect virtually every other module in the system.
3. **Interface Stability Concerns**: Any modification to the public interfaces or behavior of the extension module would potentially require corresponding changes across multiple dependent modules.
4. **Testing Burden**: Comprehensive testing would be required across the entire system to ensure changes to this core module do not introduce regressions.

The centrality of the extension module in the dependency structure means that modifications to it carry the highest risk of system-wide impacts and should be approached with appropriate caution and comprehensive testing strategies.

Risk Assessment for Module Modifications

Based on the dependency analysis, the modules with the highest risk of breaking the system if modified are:

1. **flask_sqlalchemy.extension**: As the core module with the most dependencies, modifications here have the highest potential for system-wide disruption.
2. **flask_sqlalchemy.model**: This module's involvement in cyclic dependencies with the extension module and its importation by **main** and the extension module makes it a high-risk component for modifications.
3. **flask_sqlalchemy.session**: With connections to the extension module, track_modifications module, and direct importation by **main**, changes here could have widespread effects.
4. **flask_sqlalchemy.query**: This module's usage by both the model module and the extension module, along with its own dependency on the pagination module, places it at significant risk for system impact if modified.

Lower risk modules include those with fewer inbound dependencies, such as flask_sqlalchemy.cli, flask_sqlalchemy.record_queries, and flask_sqlalchemy.table, which are only imported by **main** and the extension module but do not themselves import other system modules.

Java Class Cohesion Analysis using LCOM

Spring PetClinic: https://github.com/spring-projects/spring-petclinic

I chose the Spring PetClinic project because it serves as a well-established sample application built using the Spring framework in Java. It is designed to demonstrate common application architectures and practices, making it a realistic, yet manageable, codebase for analysis. The project is sufficiently complex to contain more than the required 10 classes, offering a good variety of components whose cohesion can be measured using LCOM. Its status as a widely recognized example also means its structure is relatively understandable, which is beneficial when analyzing LCOM results and considering potential design improvements like functional decomposition.

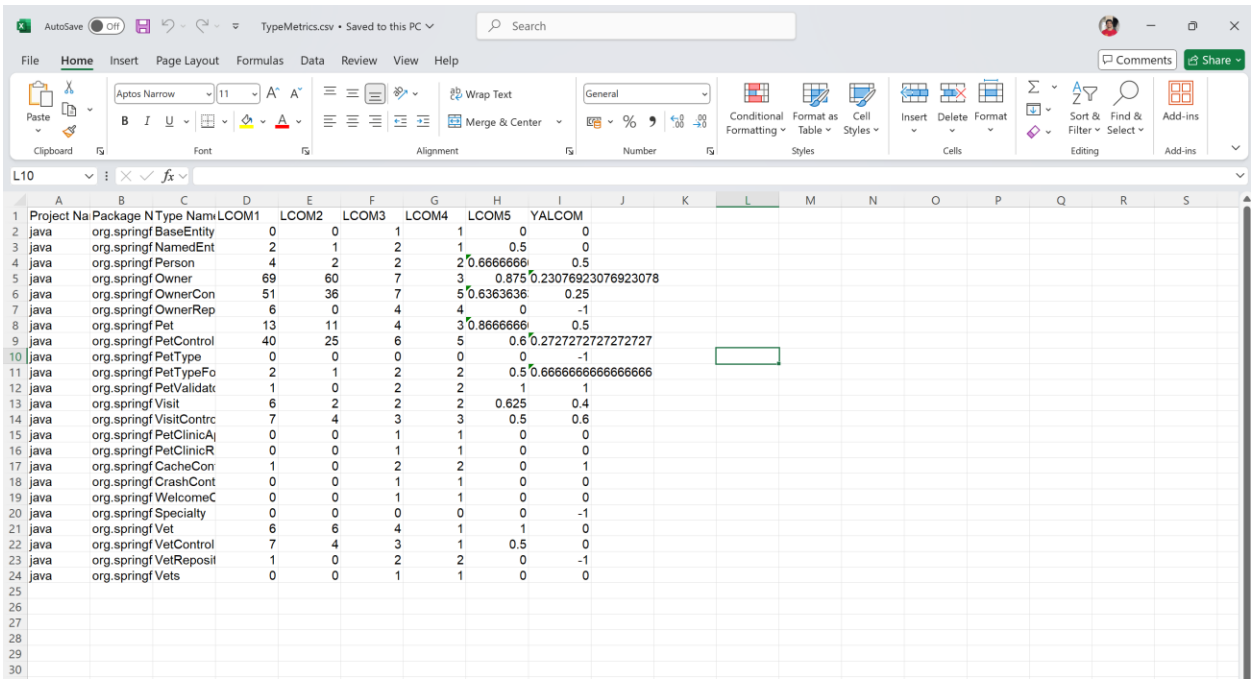
Cloning the Spring-PetClinic Repository

```
C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9>git clone https://github.com/spring-projects/spring-petclinic
Cloning into 'spring-petclinic'...
remote: Enumerating objects: 10854, done.
remote: Total 10854 (delta 0), reused 0 (delta 0), pack-reused 10854 (from 1)
Receiving objects: 100% (10854/10854), 10.23 MiB | 2.91 MiB/s, done.
Resolving deltas: 100% (4117/4117), done.
```

The LCOM.jar was downloaded and the following command was run to generate the results.

```
C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9>java -jar LCOM.jar -i "C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\spring-petclini
c\src\main\java" -o "C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\lcom-analysis"
Parsing the source code ...
Resolving symbols...
Computing metrics...
Done.
```

Output of the LCOM analysis



Project Na	Package N	Type Nam	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
java	org.springf	BaseEntity	0	0	1	1	0	0
java	org.springf	NamedEnt	2	1	2	1	0.5	0
java	org.springf	Person	4	2	2	2	0.6666666	0.5
java	org.springf	Owner	69	60	7	3	0.875	0.23076923076923078
java	org.springf	OwnerCon	51	36	7	5	0.6363636	0.25
java	org.springf	OwnerRep	6	0	4	4	0	-1
java	org.springf	Pet	13	11	4	3	0.8666666	0.5
java	org.springf	PetControl	40	25	6	5	0.6	0.2727272727272727
java	org.springf	PetType	0	0	0	0	0	-1
java	org.springf	PetTypeFo	2	1	2	2	0.5	0.6666666666666666
java	org.springf	PetValidat	1	0	2	2	1	1
java	org.springf	Visit	6	2	2	2	0.625	0.4
java	org.springf	VisitContrc	7	4	3	3	0.5	0.6
java	org.springf	PetClinicA	0	0	1	1	0	0
java	org.springf	PetClinicR	0	0	1	1	0	0
java	org.springf	CacheCon	1	0	2	2	0	1
java	org.springf	CrashCont	0	0	1	1	0	0
java	org.springf	WelcomeC	0	0	1	1	0	0
java	org.springf	Specialty	0	0	0	0	0	-1
java	org.springf	Vet	6	6	4	1	1	0
java	org.springf	VetControl	7	4	3	1	0.5	0
java	org.springf	VetReposit	1	0	2	2	0	-1
java	org.springf	Vets	0	0	1	1	0	0

The Lack of Cohesion of Methods (LCOM) metrics help us understand how well the methods and attributes in a class work together. Lower LCOM values indicate better cohesion (more connected code), while higher values suggest poor cohesion (disconnected code). The CSV file contains several LCOM variations:

- **LCOM1**: Counts pairs of methods that don't share instance variables
- **LCOM2**: Similar to LCOM1 but with slight modifications
- **LCOM3**: Measures disconnected sets of methods in a class
- **LCOM4**: Enhanced version considering method connections
- **LCOM5**: Normalized version between 0 and 1
- **YALCOM**: "Yet Another LCOM" metric for alternative measurement

Identifying Classes with High LCOM Values and Their Analysis

Looking at the provided CSV for Spring PetClinic, several classes show concerning LCOM values. The Owner class has extremely high values with LCOM1 = 69 and LCOM2 = 60, suggesting significant cohesion issues. Similarly, OwnerController (LCOM1 = 51, LCOM2 = 36) and PetController (LCOM1 = 40, LCOM2 = 25) demonstrate poor cohesion across multiple metrics.

High LCOM values generally indicate problematic class design with several implications. The class is likely trying to do too many things, violating the Single Responsibility Principle. Many methods in the class don't use shared attributes, suggesting they might belong in

different classes. These classes are typically harder to understand, maintain, and debug because their responsibilities are scattered.

For example, the Owner class with its extremely high LCOM1 (69) and LCOM2 (60) likely handles too many responsibilities related to pet owners. The high LCOM5 (0.875) confirms this issue. Similarly, Controllers in MVC frameworks often suffer from high LCOM because they coordinate multiple use cases, as seen in OwnerController and PetController which show high values across all LCOM metrics.

The PetValidator class is particularly interesting with perfect LCOM5 (1.0) and YALCOM (1.0) values, suggesting methods that operate completely independently of each other - a clear indication that the class is trying to perform unrelated validation tasks that should be separated.

Functional Decomposition Opportunities

Several classes in this project are prime candidates for functional decomposition:

1. **Owner Class**: Could be decomposed into separate classes for owner details, owner-pet relationships, and address management. The very high LCOM1 (69) suggests multiple independent responsibilities.
2. **OwnerController**: Could be split into specialized controllers for different owner operations like registration, search, and profile management.
3. **PetController**: Could be decomposed into registration, update, and type-specific controllers.
4. **PetValidator**: With perfect LCOM5 and YALCOM scores of 1.0, this class is definitely doing unrelated validations that could be separated.
5. **Vet Class**: High LCOM5 (1.0) suggests it could be split into basic vet information and specialty management.

By performing these decompositions, the codebase would likely become more maintainable, with classes that have clearer, more focused responsibilities.

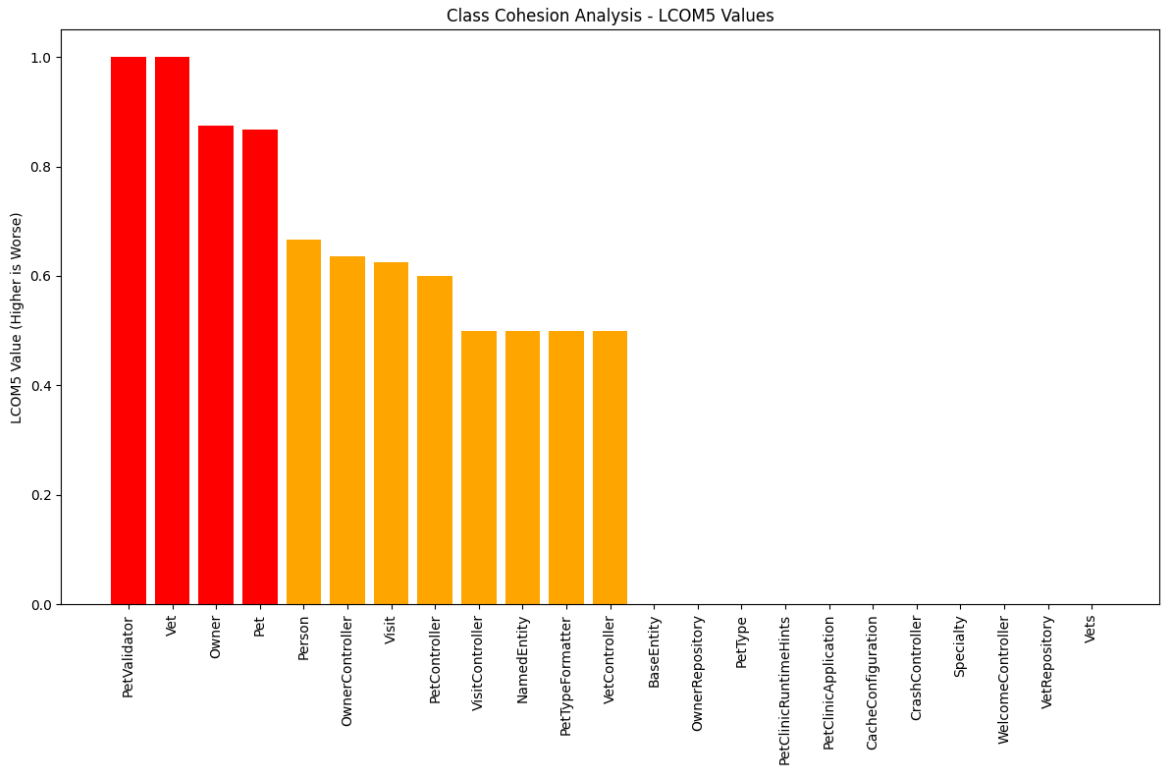
Visualizing and Analyzing Cohesion in Classes

Looking at the complete dataset, we can see different cohesion qualities across the classes:

The BaseEntity, PetType, PetClinicApplication, WelcomeController, and Specialty classes demonstrate excellent cohesion with very low LCOM values. These classes are likely focused on doing one thing well, following the Single Responsibility Principle effectively.

In contrast, classes like NamedEntity, Person, Visit, and VisitController show moderate cohesion issues that might benefit from some refactoring but aren't critical problems.

The most troubling classes are Owner, OwnerController, Pet, PetController, PetValidator, and Vet, which show poor cohesion across multiple LCOM metrics. Approximately 26% of the classes in the system show significant cohesion problems, with another 22% showing moderate issues. This suggests systematic design problems that might benefit from architectural refactoring.



A script lcom-analysis.py was also written to summarise the findings from the csv file.

```
(base) C:\Users\srira\OneDrive - iitgn.ac.in\STT for CS\Lab 9\lcom-analysis>python lcom-analysis.py
=== LCOM Cohesion Analysis for Spring PetClinic ===

Total classes analyzed: 23

Cohesion Quality Distribution:
- Good: 11 classes (47.8%)
- Moderate: 8 classes (34.8%)
- Poor: 4 classes (17.4%)

Top 5 Classes with Poorest Cohesion (Highest LCOM5):
- PetValidator: LCOM5=1.000, LCOM1=1.0
- Vet: LCOM5=1.000, LCOM1=6.0
- Owner: LCOM5=0.875, LCOM1=69.0
- Pet: LCOM5=0.867, LCOM1=13.0
- Person: LCOM5=0.667, LCOM1=4.0

Classes with Perfect LCOM5 (1.0) - Immediate Refactoring Candidates:
- PetValidator
- Vet

Analysis chart saved as 'lcom_analysis.png'

Detailed Recommendations for Top 3 Problematic Classes:

PetValidator (LCOM5: 1.000):
Potential issues:
- Methods operating independently with minimal attribute sharing
Recommendation:
- Perform class-level refactoring to increase method-attribute cohesion
- Consider applying Single Responsibility Principle more strictly

Vet (LCOM5: 1.000):
Potential issues:
- Methods operating independently with minimal attribute sharing
Recommendation:
- Extract specialized classes for different aspects of the entity
- Consider using composition instead of putting everything in one class

Owner (LCOM5: 0.875):
Potential issues:
- Extremely high number of method pairs not sharing attributes
- Methods operating independently with minimal attribute sharing
Recommendation:
- Extract specialized classes for different aspects of the entity
- Consider using composition instead of putting everything in one class
```

	Package Na	Type Name	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
		(name = "owners") public class Owner extends Person {						
		@Column(name = "address") @NotBlank private String address;						
		@Column(name = "city") @NotBlank private String city;						
		@Column(name = "telephone") @NotBlank @Pattern(regexp = "\\d{10}", message = "Telephone must be a 10-digit number") private String telephone;						
	org.springfi	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER) @JoinColumn(name = "owner_id")	69	60	7	3	0.875	0.23076923076923078

Side-by-side comparison can be found in FilteredMetrics.csv.

Results and Analysis

LCOM analysis of the Spring PetClinic

The LCOM analysis of the Spring PetClinic project reveals several classes with significant cohesion issues. The primary candidates for refactoring include Owner, OwnerController,

PetController, and PetValidator classes, all of which show signs of trying to do too many things at once.

To improve the design, the controllers should be decomposed into more focused components that handle specific use cases. Entity classes like Owner and Pet should possibly be broken down using composition rather than putting all behavior in a single class. Validator classes should follow the single responsibility principle more strictly.

This analysis highlights how LCOM metrics can effectively identify design issues in a Java project, helping developers make informed decisions about when and how to refactor their code for better maintainability and reusability. By addressing these cohesion issues, the Spring PetClinic project could become more modular, easier to understand, and simpler to maintain over time.

Discussion and Conclusion

The analysis of software module dependencies using pydeps and class cohesion using LCOM has provided valuable insights into the structural health of the two selected projects: Flask-SQLAlchemy and Spring PetClinic. This comprehensive examination reveals important patterns in software architecture that impact maintainability, extensibility, and overall code quality.

For the Flask-SQLAlchemy project, the dependency analysis revealed a centralized architecture with flask_sqlalchemy.extension serving as a hub module with high coupling. The presence of cyclic dependencies between key modules like flask_sqlalchemy ↔ flask_sqlalchemy.extension creates potential maintenance challenges. These circular references make the system more rigid and less amenable to isolated changes, as modifications to one component necessitate careful consideration of the entire dependency cycle.

Similarly, the cohesion analysis of Spring PetClinic using LCOM metrics identified several classes with poor cohesion, particularly Owner, OwnerController, PetController, and PetValidator. These classes attempt to handle too many responsibilities, violating the Single Responsibility Principle and creating maintenance challenges. High LCOM values indicate methods operating independently with little shared state, suggesting these classes are prime candidates for functional decomposition.

Both analyses demonstrate how automated tooling can help identify architectural issues that might otherwise go unnoticed until they become problematic during maintenance or

expansion phases. The visualization of dependencies and quantification of cohesion provide objective measures that complement subjective code reviews.

- Automated analysis tools serve as effective early warning systems for design issues that could impact long-term maintainability.
- Both Flask-SQLAlchemy and Spring PetClinic, despite being well-established projects, contain structural issues that could benefit from targeted refactoring.
- Architectural patterns influence cohesion and coupling measurements, with MVC frameworks like Spring showing characteristic patterns in their controllers and models.
- Quantitative metrics provide an objective basis for refactoring decisions, helping development teams prioritize improvements with the highest impact.
- Regular application of these analysis tools throughout the development lifecycle could help prevent architectural degradation and maintain software quality over time.

CS202: Software Tools and Techniques for CSE

Lab Assignment 10 (27 March 2025)

By Sriram Srinivasan (22110258)

Introduction, Setup, and Tools

Overview

This lab focused on the development of C# console applications using the Visual Studio 2022 IDE within the .NET environment. The aim was to gain hands-on experience with C# programming by implementing basic control structures, functions, object-oriented concepts, exception handling, and debugging techniques. Each activity was designed to build practical understanding and improve programming fluency using professional development tools.

Objectives

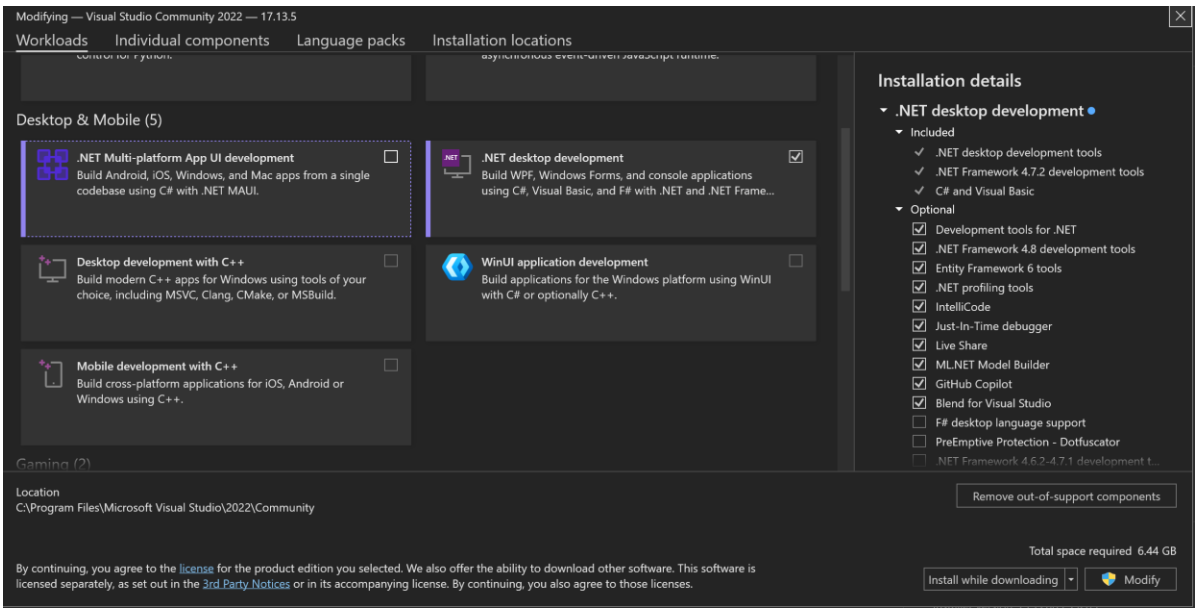
- To set up and use Visual Studio for .NET development using C#.
- To write and execute basic console applications.
- To implement control structures like loops and conditionals.
- To understand and apply object-oriented programming (OOP) in C#.
- To utilize the Visual Studio Debugger for step-by-step program execution.
- To handle exceptions gracefully in user input and calculations.

Environment Setup and Tool Versions Used

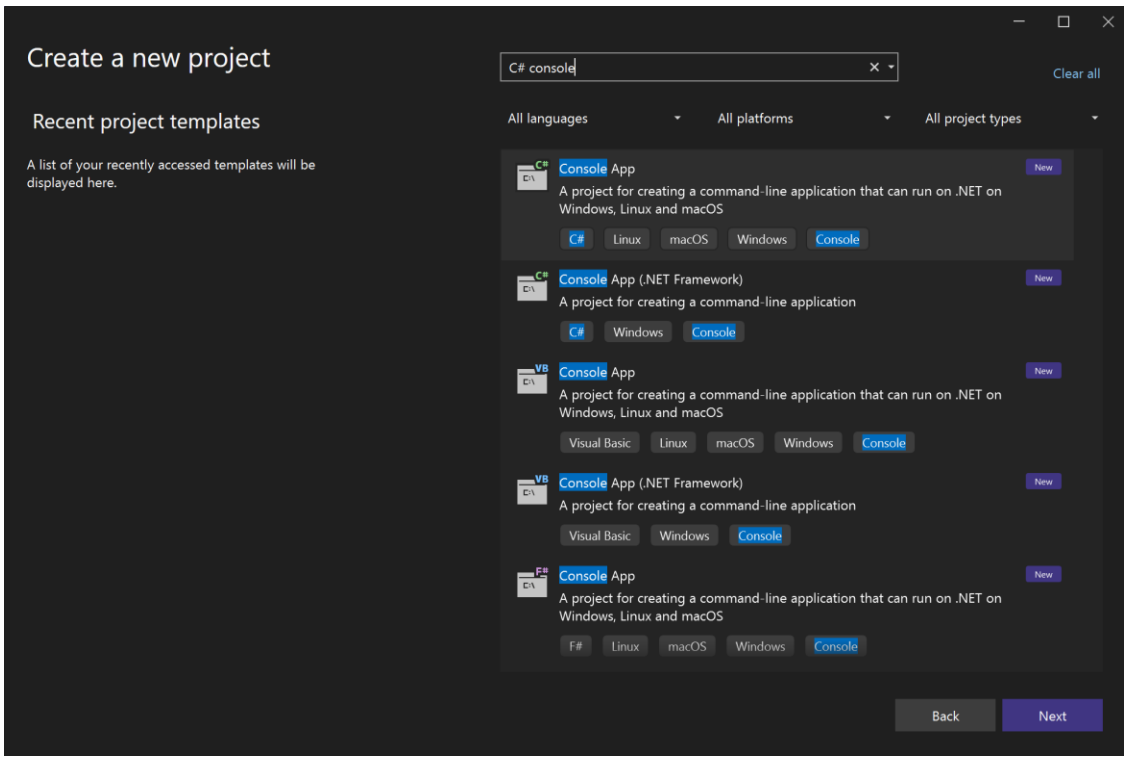
- **Operating System**: Windows 10 / Windows 11
- **IDE**: Visual Studio 2022 (Community Edition)
- **.NET SDK**: .NET 6.0 (or higher)
- **Programming Language**: C# (Latest stable version)
- **Debugger**: Visual Studio Debugger (Breakpoints, Step-In, Step-Over, Step-Out)

Methodology and Execution

Activity 1: Setting Up .NET Development Environment



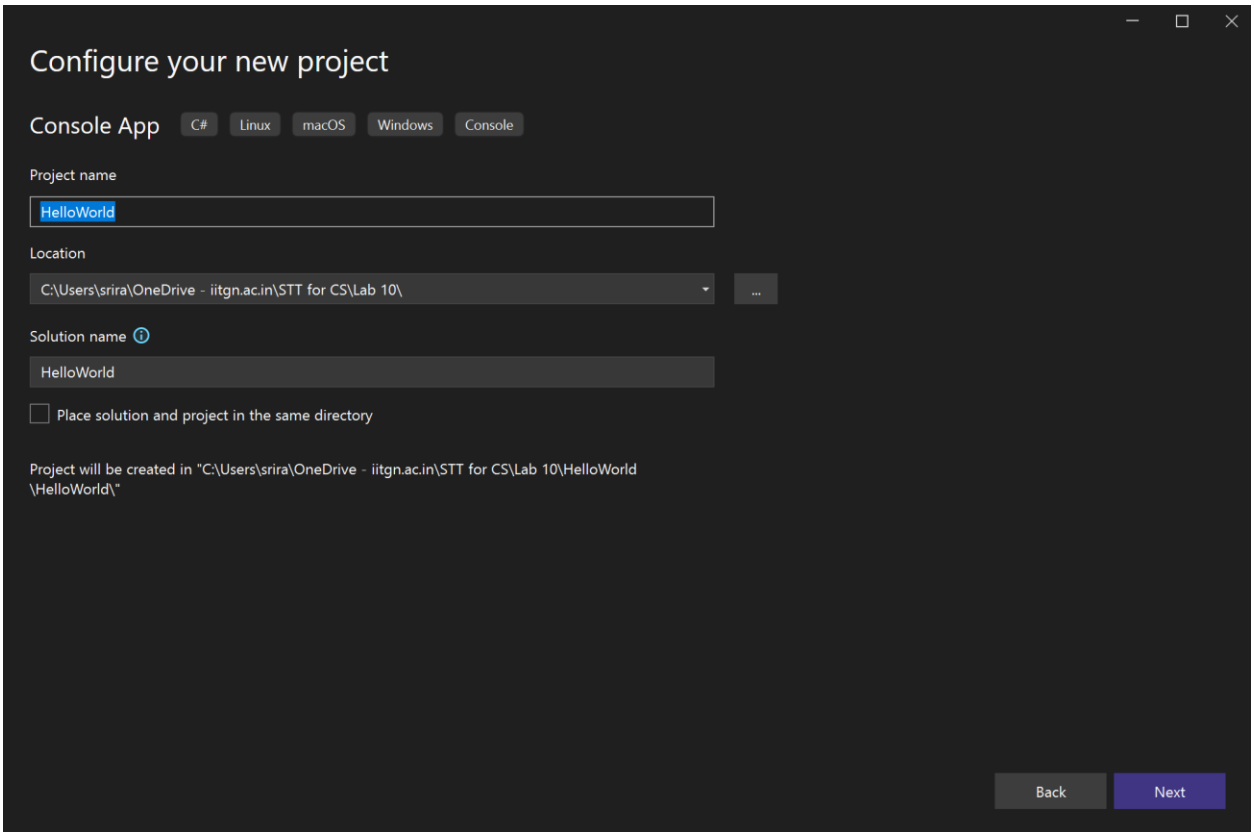
Installation of the .NET desktop environment on Visual Studio



Creating a new project (C# Console App with .NET 6+)

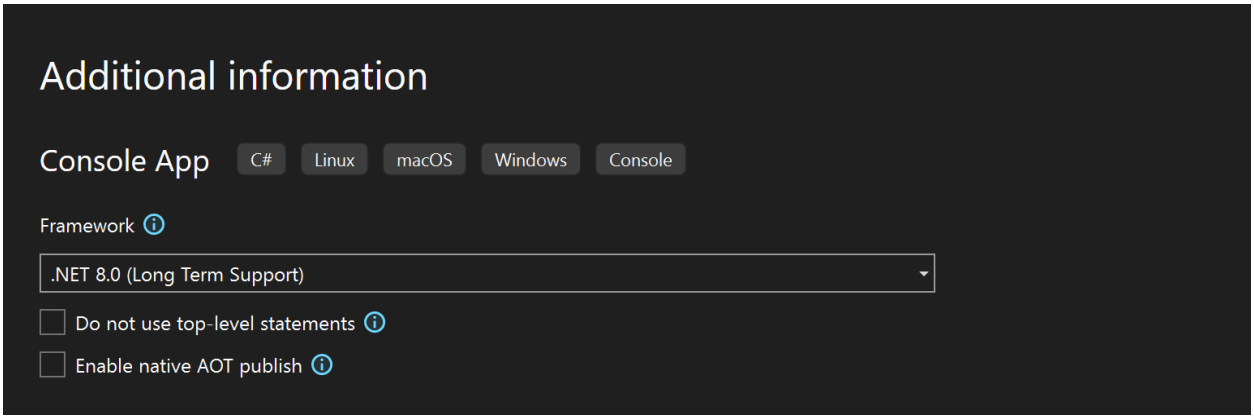
The key difference between .NET and .NET Framework projects lies in platform compatibility and modern development support. .NET (Core/6/7/8) is cross-platform, faster, and ideal for new console applications, while .NET Framework is Windows-only and suited for maintaining legacy applications. For this lab, using .NET 6 or later is recommended to align with current C# development practices.

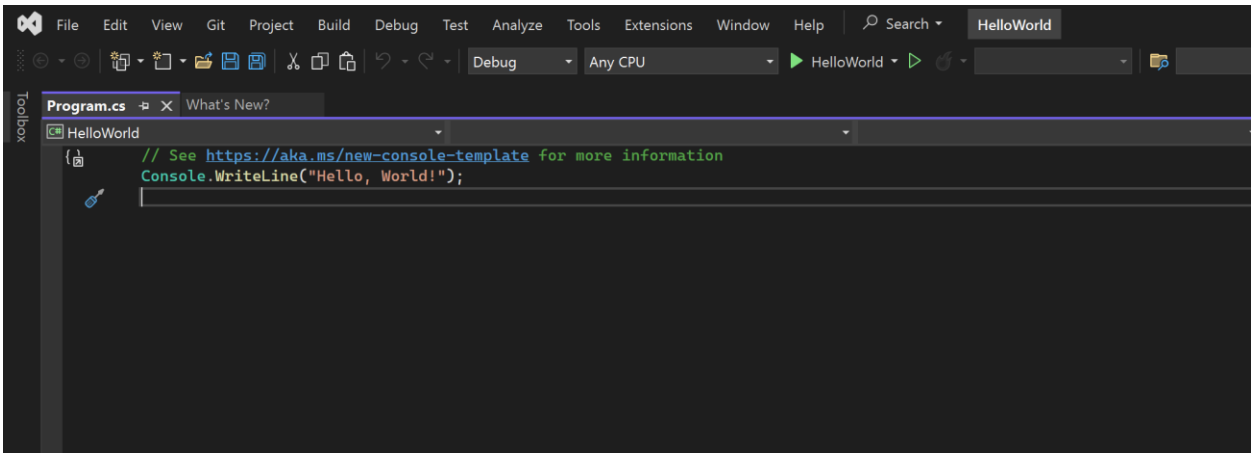
A sample project is created to check that all installations have been correctly made.



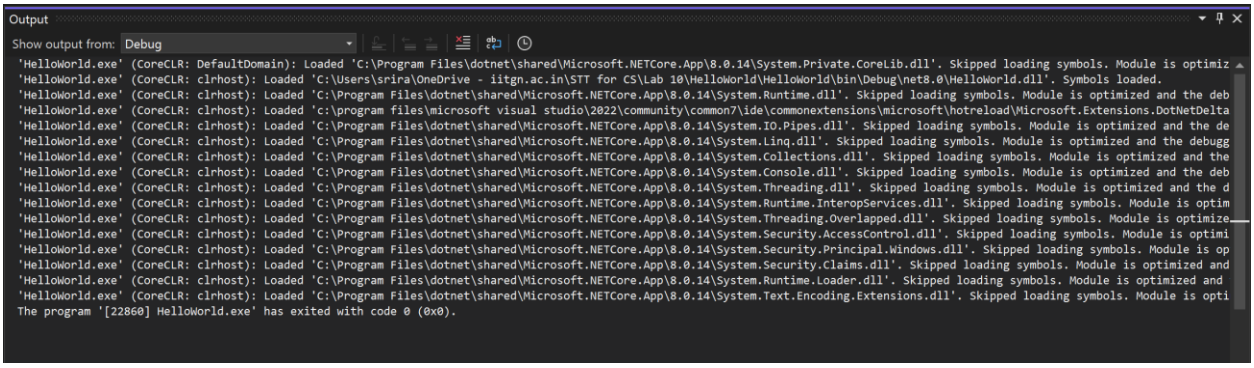
Creation of HelloWorld project

The installed dotnet SDK version is 8.0.14.





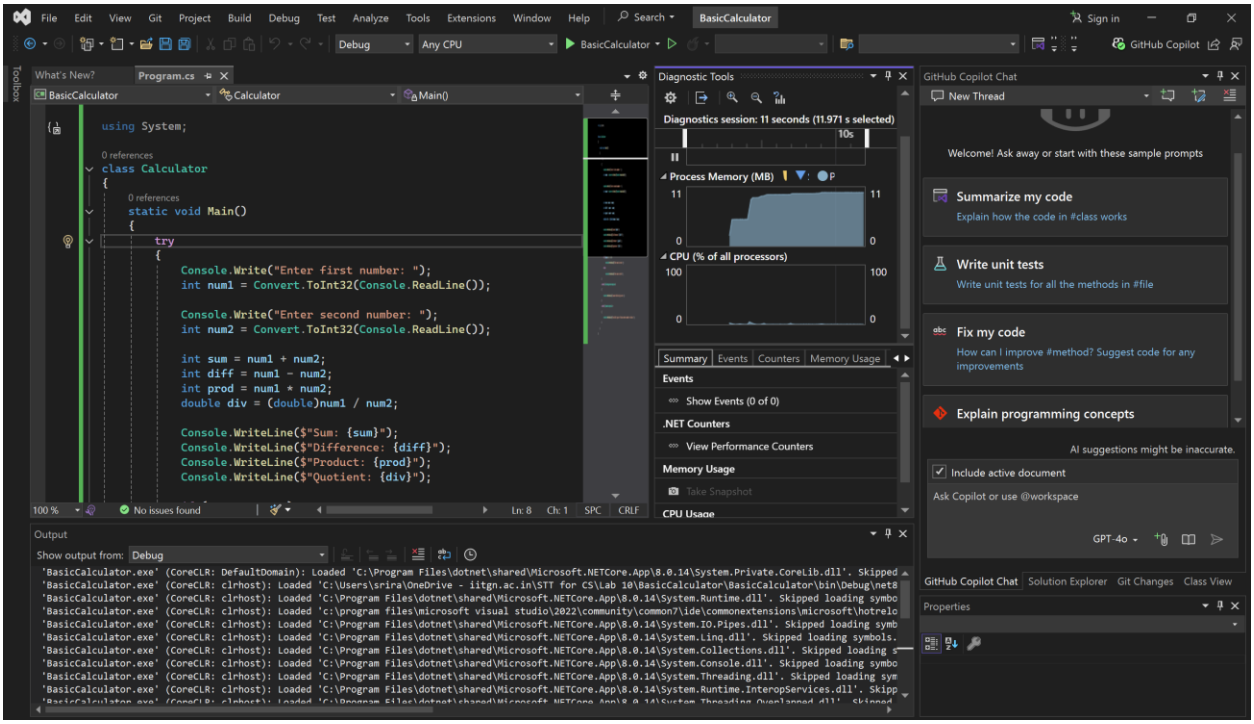
Code for printing ”Hello World” to the console



Activity 2: Understanding Basic Syntax and Control Structures

A console application was developed to accept two numeric inputs from the user. After converting the inputs to integers, basic

arithmetic operations—addition, subtraction, multiplication, and division—were performed. The sum was evaluated using conditional statements to determine whether it was even or odd. All outputs were printed using the `Console.WriteLine()` method. This activity provided practical exposure to input handling, data conversion, conditionals, and output formatting in C#.

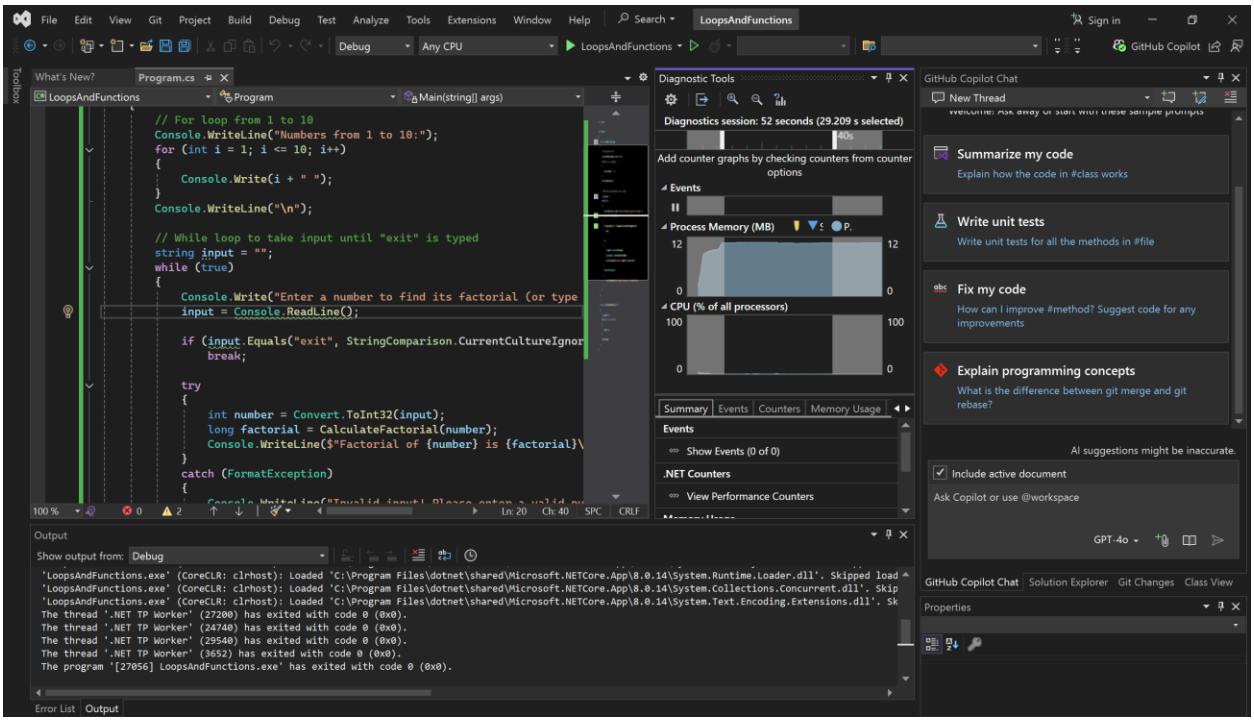


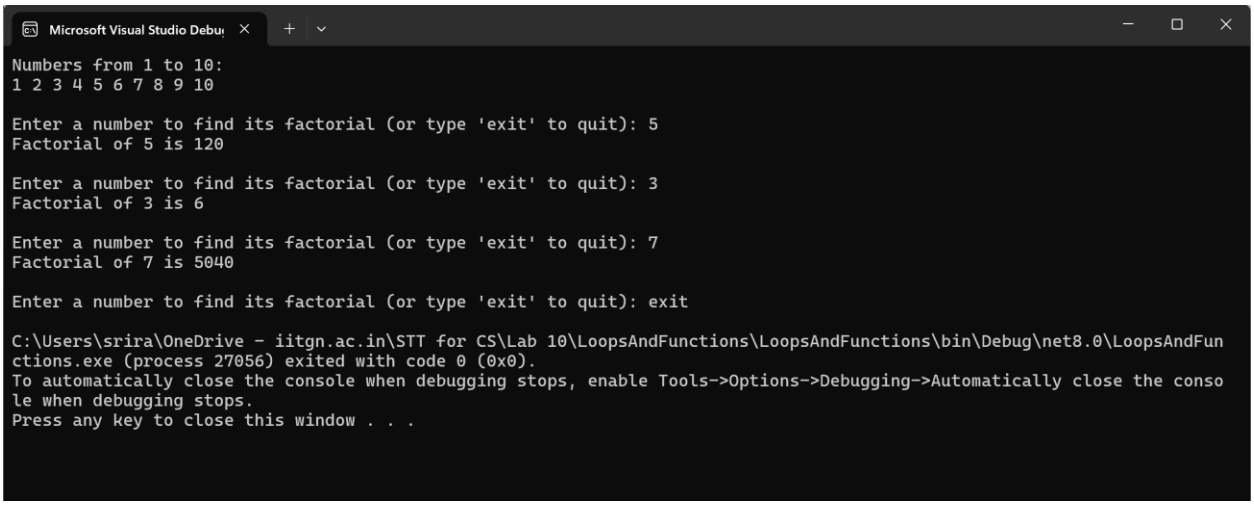
Analysis window after the execution of BasicCalculator

Output of BasicCalculator

Activity 3: Implementing Loops and Functions

To demonstrate iteration and modular programming, a program was designed with three components. A `for` loop was used to print numbers from 1 to 10, showcasing fixed iteration. A `while` loop was introduced to repeatedly prompt for input until the user typed “exit”, illustrating a user-controlled loop. A separate function was defined to calculate the factorial of a given number, promoting code reuse and structure. This activity helped reinforce understanding of control flow and the use of functions in building readable, maintainable code.

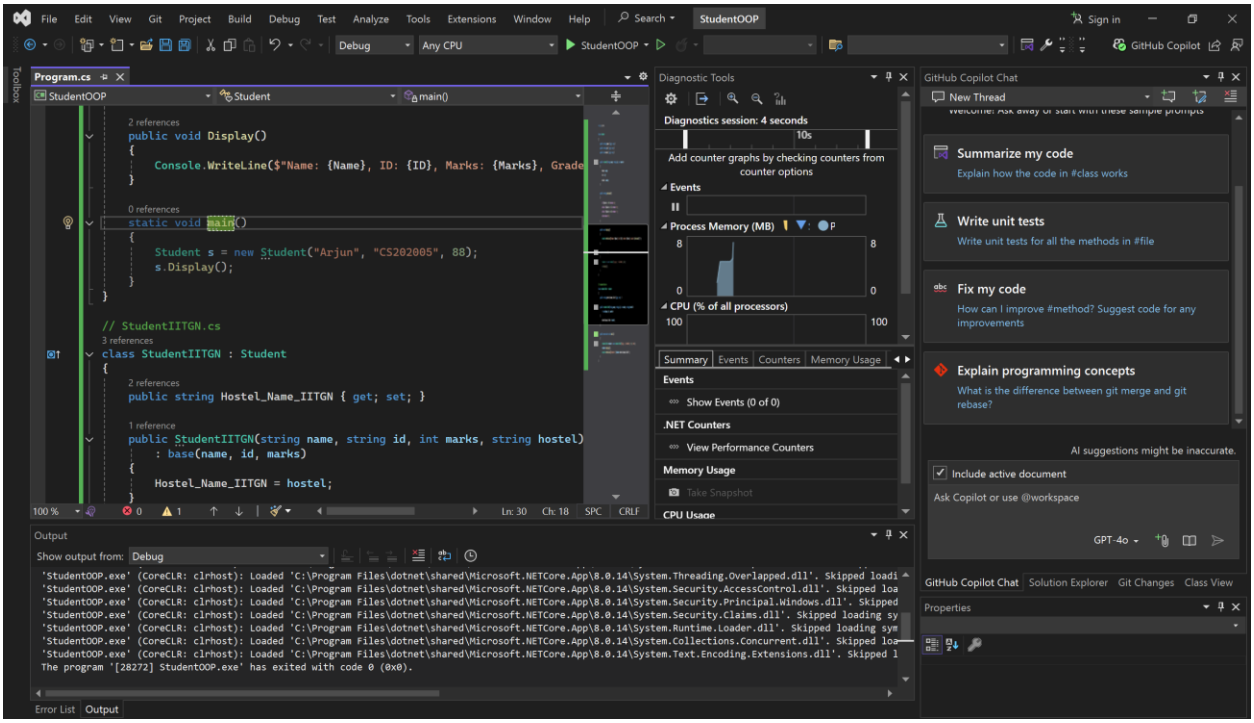


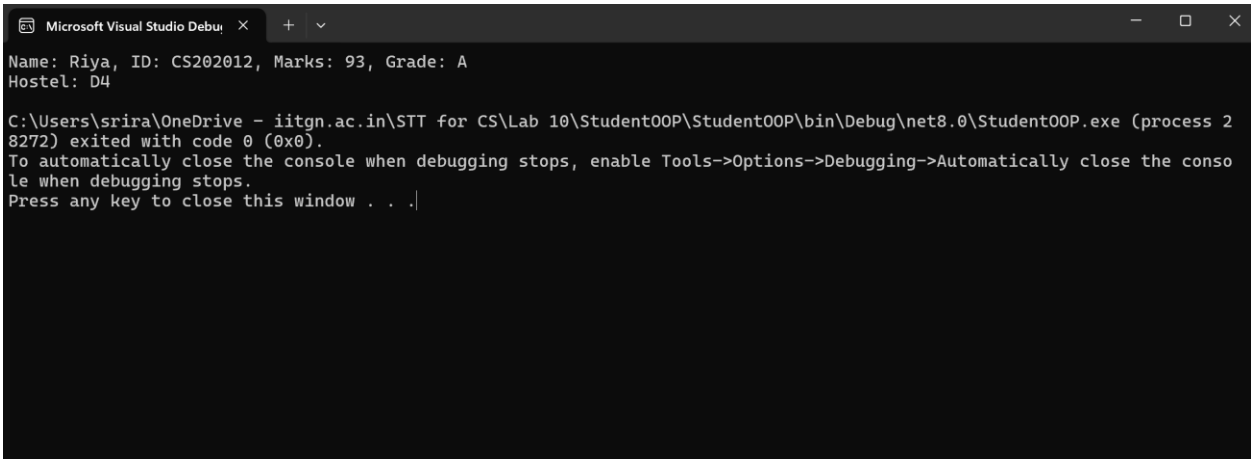


Activity 4: Perform Object-Oriented Programming in C#

A class `Student` was created with properties for name, ID, and marks, alongside a constructor for initialization and a method `GetGrade()` to return a letter grade based on marks. To apply inheritance, a subclass `StudentIITGN` extended the base class and introduced an additional property for hostel name. Object instances were created in the `Main()` method to display student

information. This activity illustrated the practical application of OOP principles such as encapsulation, inheritance, and constructor usage within C#.



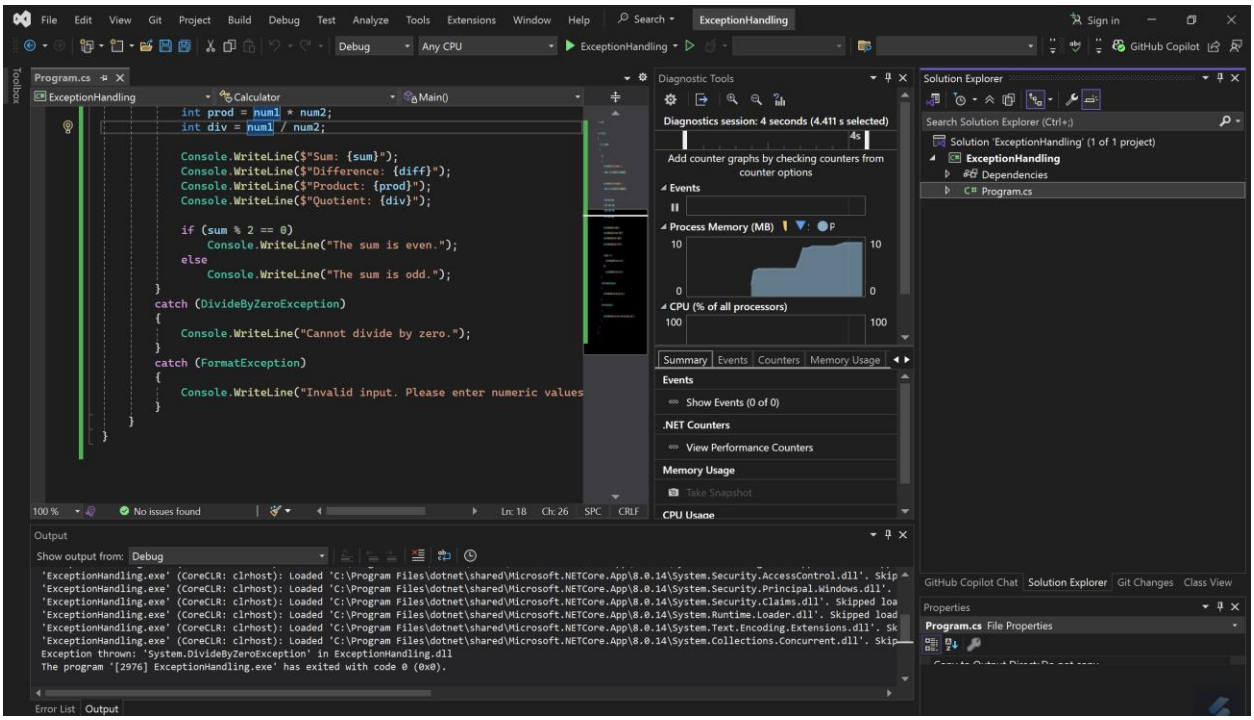


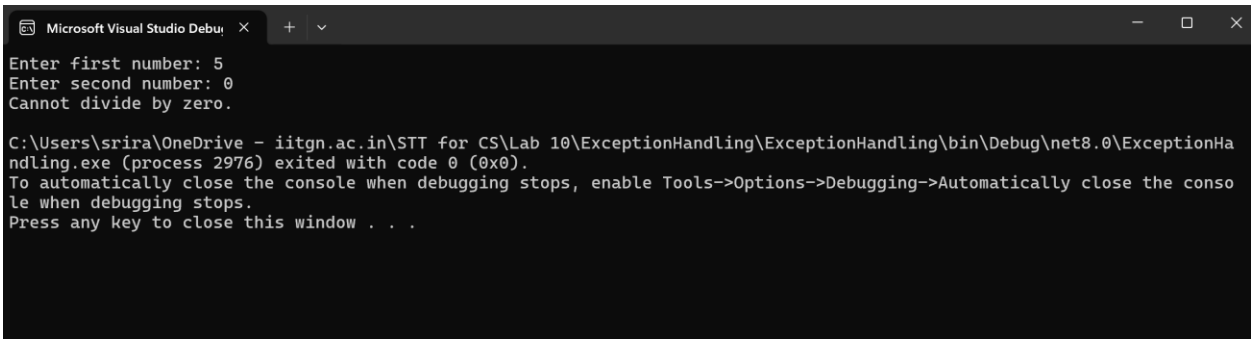
The multiple entry point issue in Activity 4 was resolved by renaming one of the `Main()` methods to lowercase (`main()`), which made it unrecognized as a valid entry point by the C# compiler. This effectively left only one valid `Main()` method in the project, allowing the program to compile and run successfully. Although functional, a

more structured approach would be to maintain a single `Main()` and route all class executions through it.

Activity 5: Exception Handling

The calculator program was enhanced to handle runtime errors through structured exception handling. Input and arithmetic operations were enclosed within a `try-catch` block. Specific exceptions such as `DivideByZeroException` and `FormatException` were caught to manage invalid inputs and division errors. The program was able to recover gracefully from errors without crashing, demonstrating the use of robust error-handling techniques to ensure application stability.





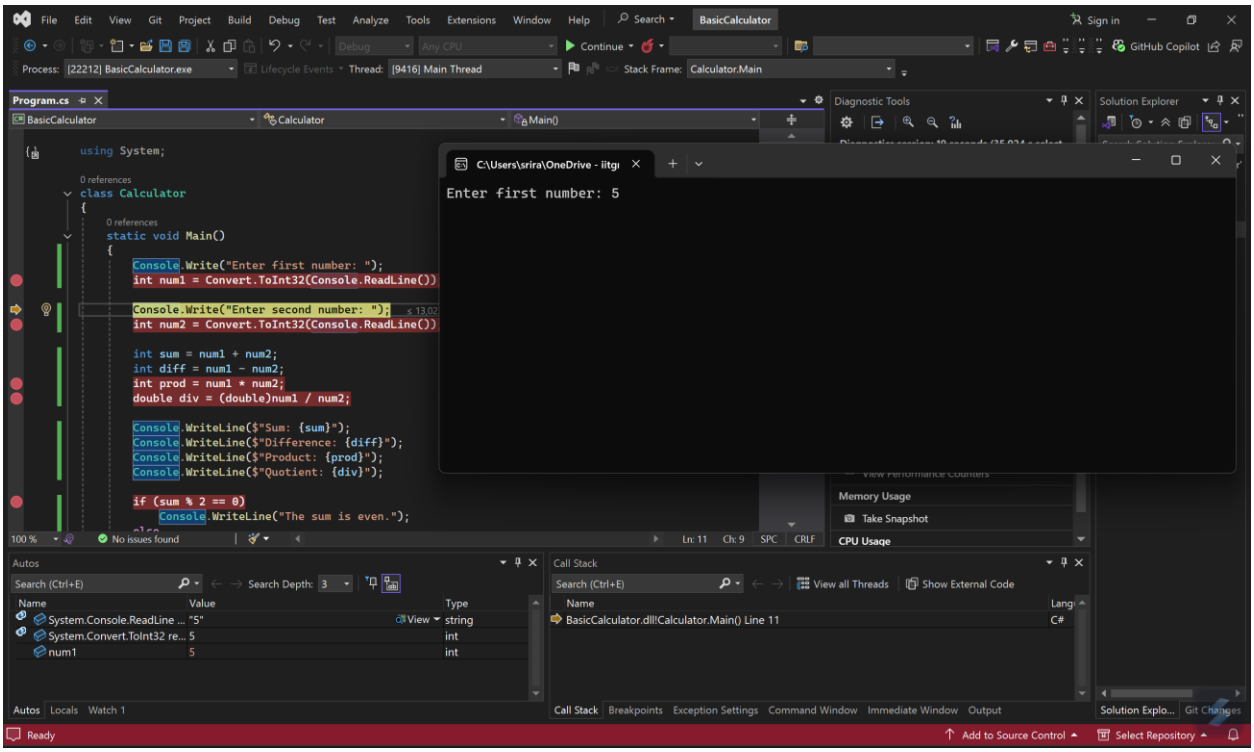
Activity 6: Debugging using Visual Studio Debugger

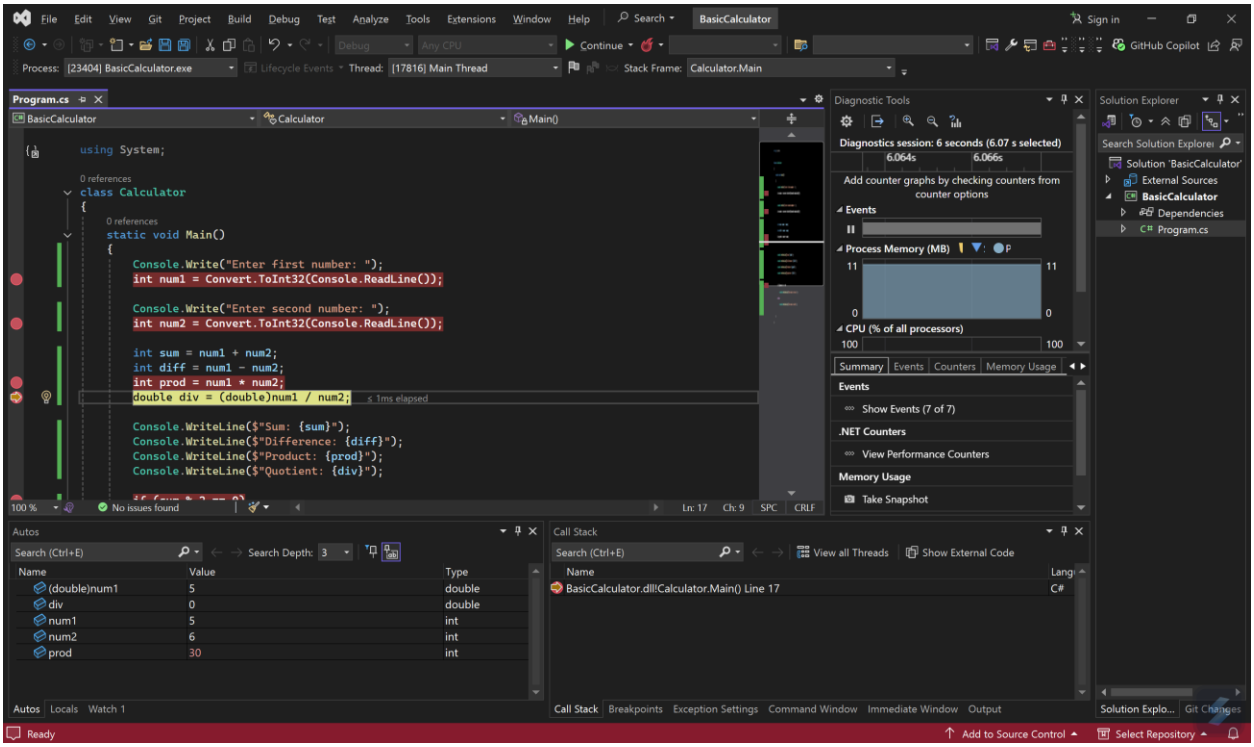
During the debugging phase of this lab, breakpoints were added at key logical points across Activities 2 to 5 to closely observe the program's execution flow and identify how the control moved through various parts of the code. In Activity 2, breakpoints were placed at the points where user inputs were taken and where arithmetic operations were performed. This helped track the input values and verify if the control structures like `if-else` conditions were being correctly evaluated. For Activity 3, additional breakpoints were set inside the `for` loop, `while` loop, and at the start of the factorial function. These helped in watching how loops behaved during iteration and how the function executed step by step.

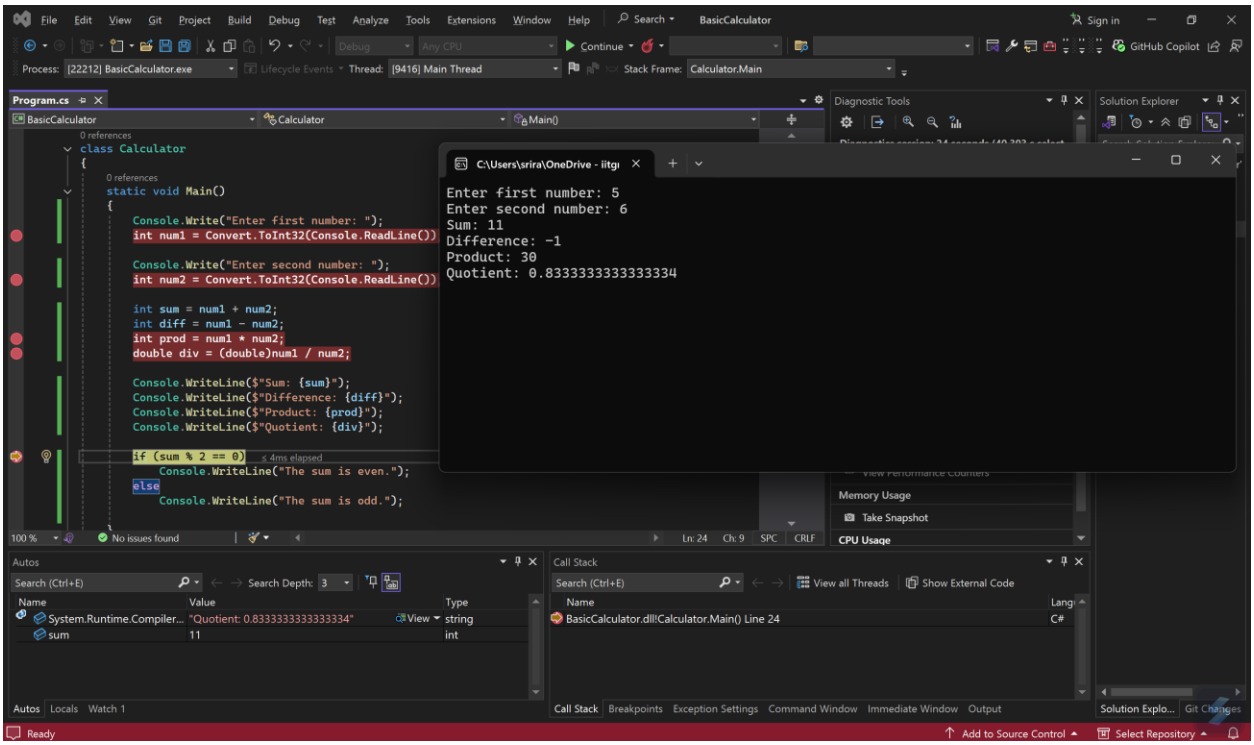
In Activity 4, which focused on object-oriented programming, breakpoints were placed inside the constructors, property initializations, and methods like `getGrade()` to observe how data members were assigned and how inheritance worked when the subclass was used. Finally, in Activity 5, breakpoints were mainly

added around the potentially risky operations, especially before division, and inside the `catch` blocks to monitor how exceptions were triggered and handled. The debugger tools like Step-In, Step-Over, and Step-Out were selectively used while navigating into custom methods, skipping known functions, or exiting from within method calls. Overall, strategic breakpoint placement allowed a clear, step-by-step examination of logic and helped in better understanding how the program transitioned from one block to another.

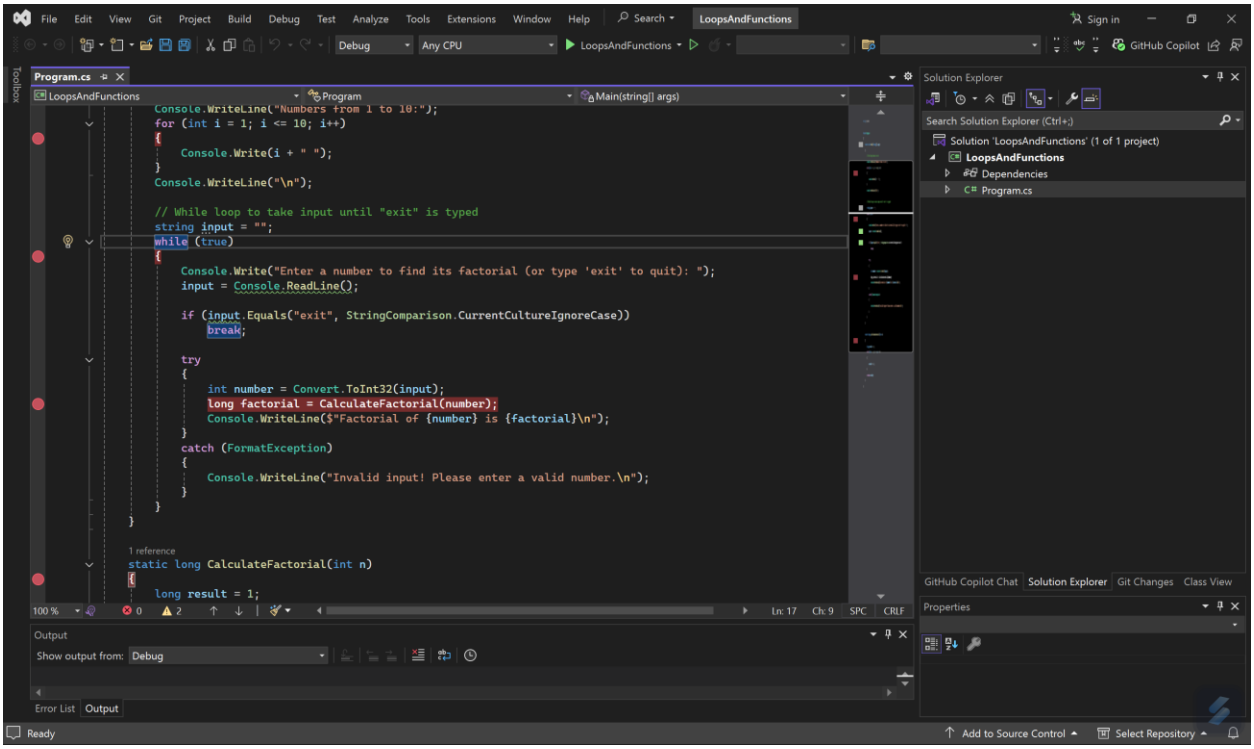
Activity 2

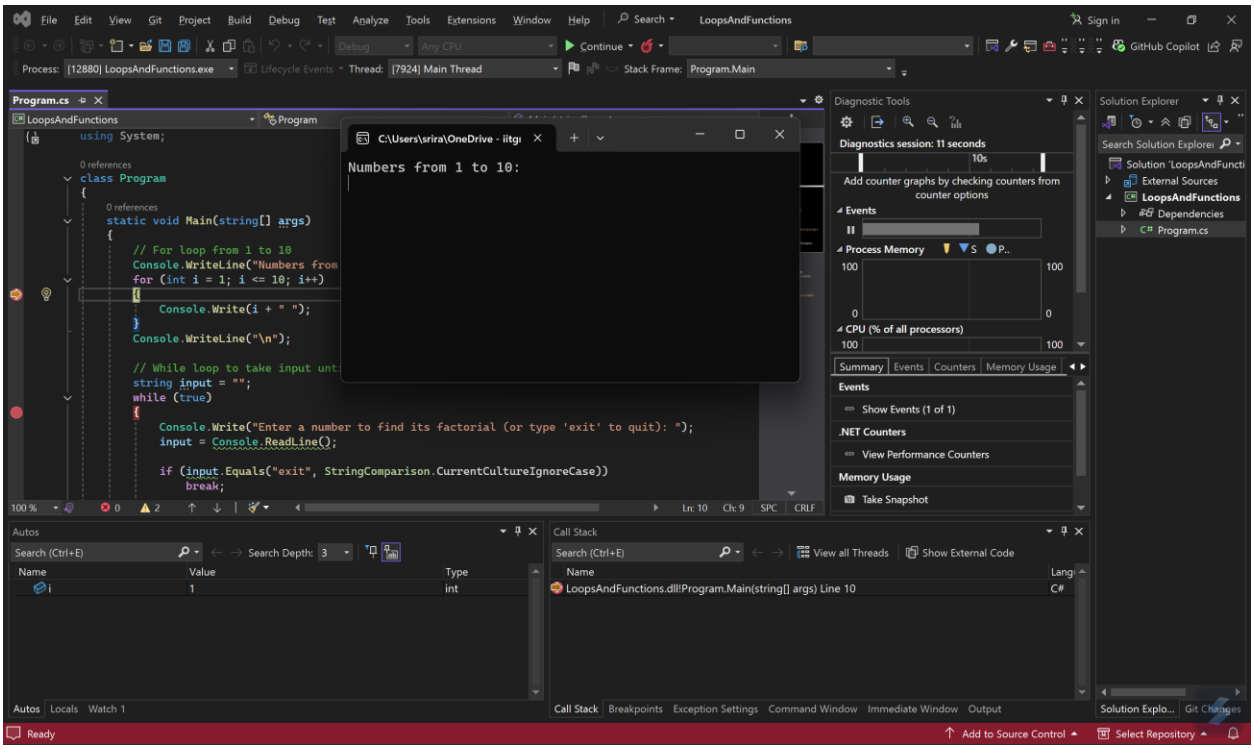

Program.cs
using System;
class Calculator
{
static void Main()
{
Console.Write("Enter first number: ");
int num1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter second number: ");
int num2 = Convert.ToInt32(Console.ReadLine());
int sum = num1 + num2;
int diff = num1 - num2;
int prod = num1 * num2;
double div = (double)num1 / num2;
Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Difference: {diff}");
Console.WriteLine($"Product: {prod}");
Console.WriteLine($"Quotient: {div}");
if (sum % 2 == 0)
Console.WriteLine("The sum is even.");
C:\Users\srira\OneDrive - iitgn
Enter first number: 5
Autos
Name Value Type
System.Console.ReadLine ... "5" string
System.Convert.ToInt32 re... 5 int
num1 5 int
Call Stack
BasicCalculator.dll!Calculator.Main() Line 11
Ready

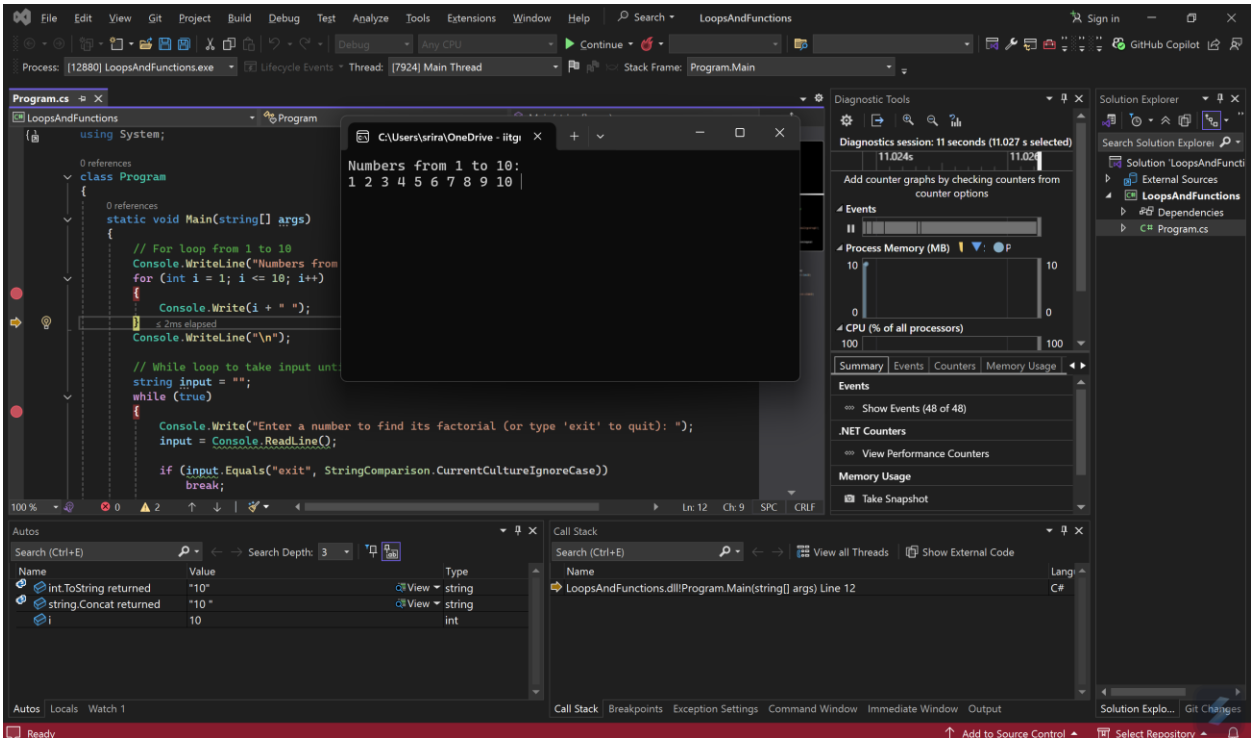

Program.cs
using System;
class Calculator
{
static void Main()
{
Console.Write("Enter first number: ");
int num1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter second number: ");
int num2 = Convert.ToInt32(Console.ReadLine());
int sum = num1 + num2;
int diff = num1 - num2;
int prod = num1 * num2;
double div = (double)num1 / num2;
Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Difference: {diff}");
Console.WriteLine($"Product: {prod}");
Console.WriteLine($"Quotient: {div}");
Diagnostic Tools
Diagnostics session: 6 seconds (6.07 s selected)
Autos
Name Value Type
(double)num1 5 double
div 0 double
num1 5 int
num2 6 int
prod 30 int
Call Stack
BasicCalculator.dll!Calculator.Main() Line 17
Ready

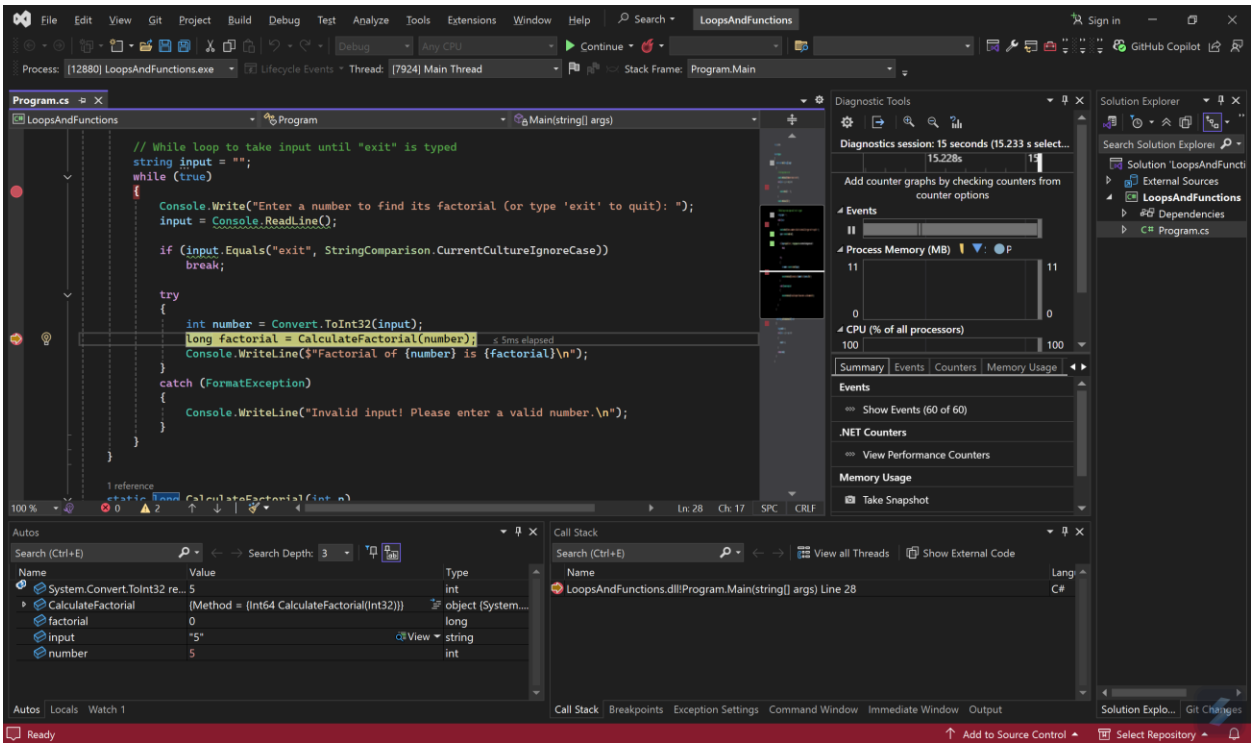

Program.cs
BasicCalculator
Calculator
Main()
class Calculator
{
static void Main()
{
Console.Write("Enter first number: ");
int num1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter second number: ");
int num2 = Convert.ToInt32(Console.ReadLine());
int sum = num1 + num2;
int diff = num1 - num2;
int prod = num1 * num2;
double div = (double)num1 / num2;
Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Difference: {diff}");
Console.WriteLine($"Product: {prod}");
Console.WriteLine($"Quotient: {div}");
if (sum % 2 == 0)
Console.WriteLine("The sum is even.");
else
Console.WriteLine("The sum is odd.");
C:\Users\srira\OneDrive - iitgı
Enter first number: 5
Enter second number: 6
Sum: 11
Difference: -1
Product: 30
Quotient: 0.8333333333333334
Memory Usage
Take Snapshot
CPU Usage
Autos
Name
Value
Type
System.Runtime.Compiler...
"Quotient: 0.8333333333333334"
string
sum
11
int
Call Stack
BasicCalculator.dll!Calculator.Main() Line 24
Ready

Activity 3

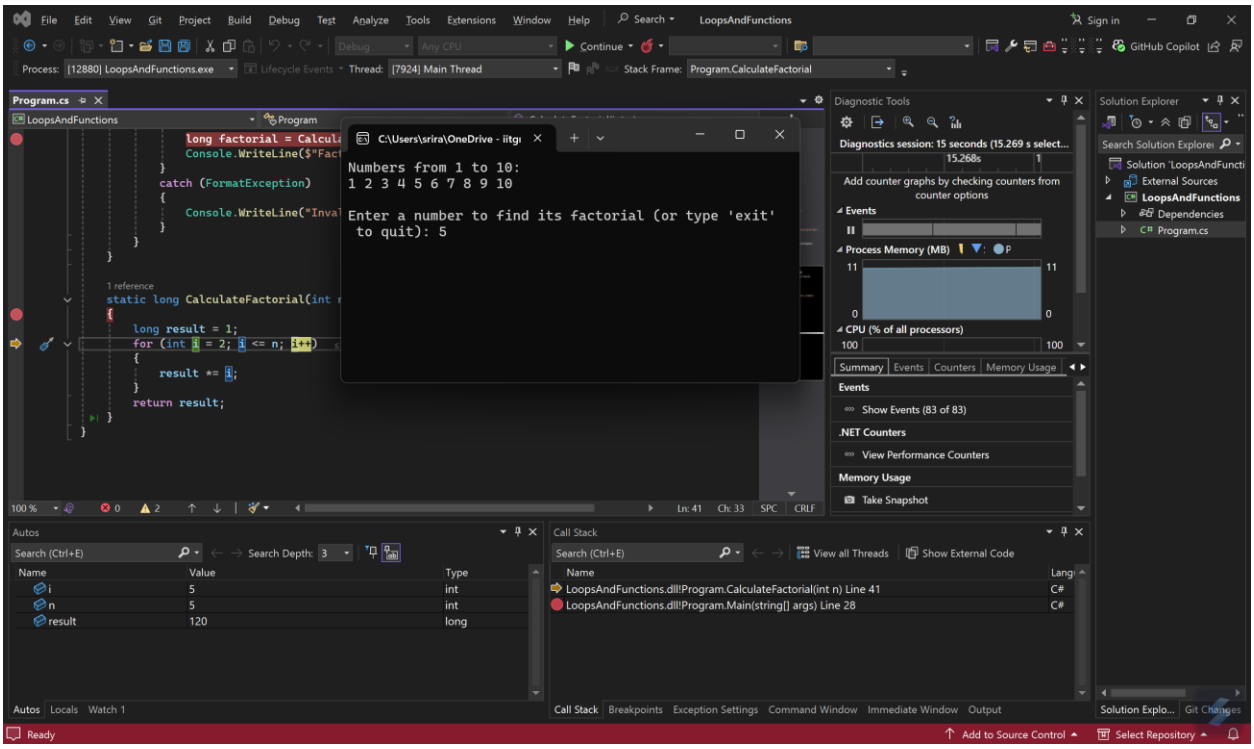

Program.cs
LoopsAndFunctions
Program
Main(string[] args)
Console.WriteLine("Numbers from 1 to 10:");
for (int i = 1; i <= 10; i++)
{
Console.Write(i + " ");
}
Console.WriteLine("\n");
// While loop to take input until "exit" is typed
string input = "";
while (true)
{
Console.Write("Enter a number to find its factorial (or type 'exit' to quit): ");
input = Console.ReadLine();
if (input.Equals("exit", StringComparison.CurrentCultureIgnoreCase))
break;
try
{
int number = Convert.ToInt32(input);
long factorial = CalculateFactorial(number);
Console.WriteLine($"Factorial of {number} is {factorial}\n");
}
catch (FormatException)
{
Console.WriteLine("Invalid input! Please enter a valid number.\n");
}
}
}
1 reference
static long CalculateFactorial(int n)
{
long result = 1;
Solution Explorer
Solution 'LoopsAndFunctions' (1 of 1 project)
LoopsAndFunctions
Dependencies
Program.cs
Output
Show output from: Debug
Error List
Output
Ready

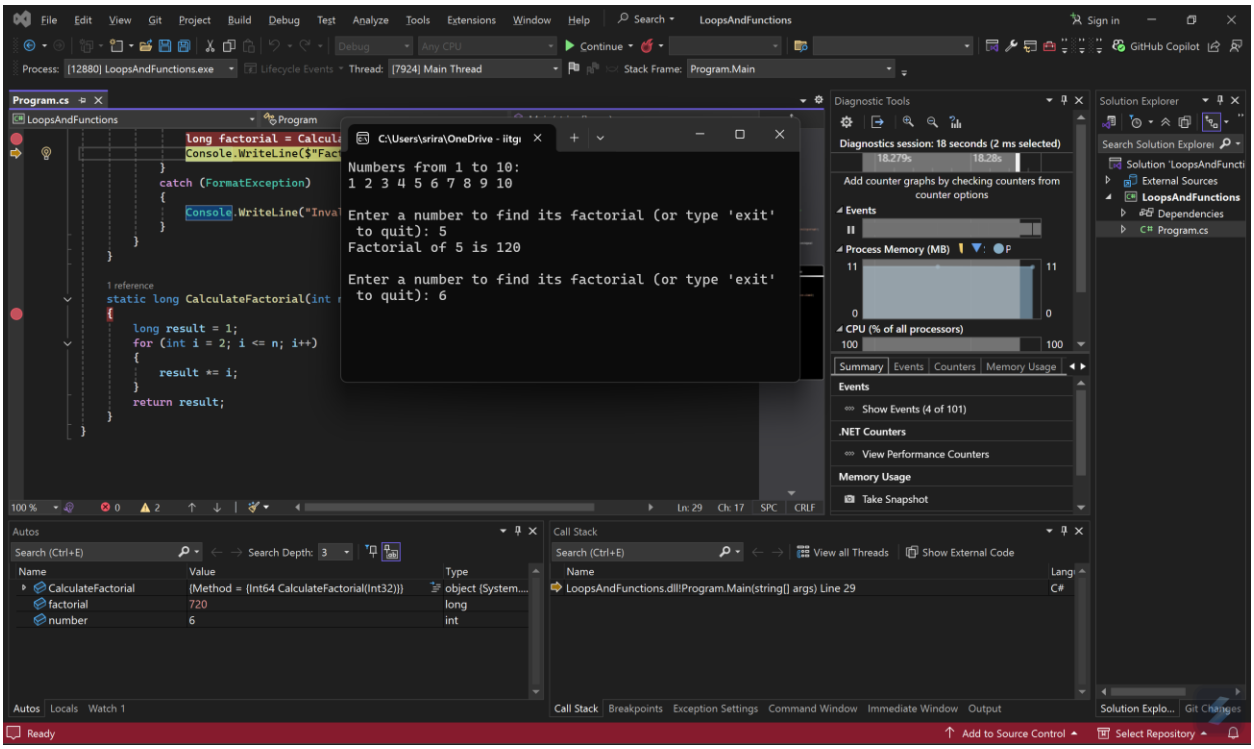

Numbers from 1 to 10:
Enter a number to find its factorial (or type 'exit' to quit):


Numbers from 1 to 10:
1 2 3 4 5 6 7 8 9 10
Enter a number to find its factorial (or type 'exit' to quit):

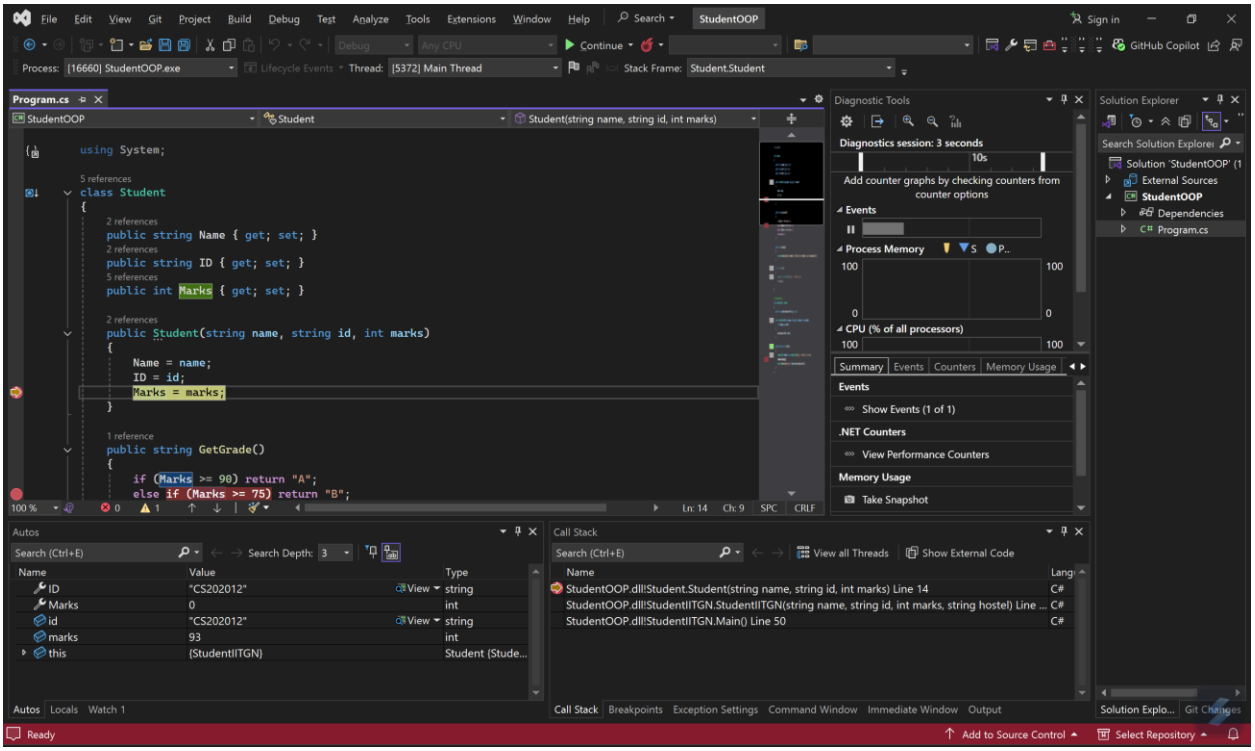


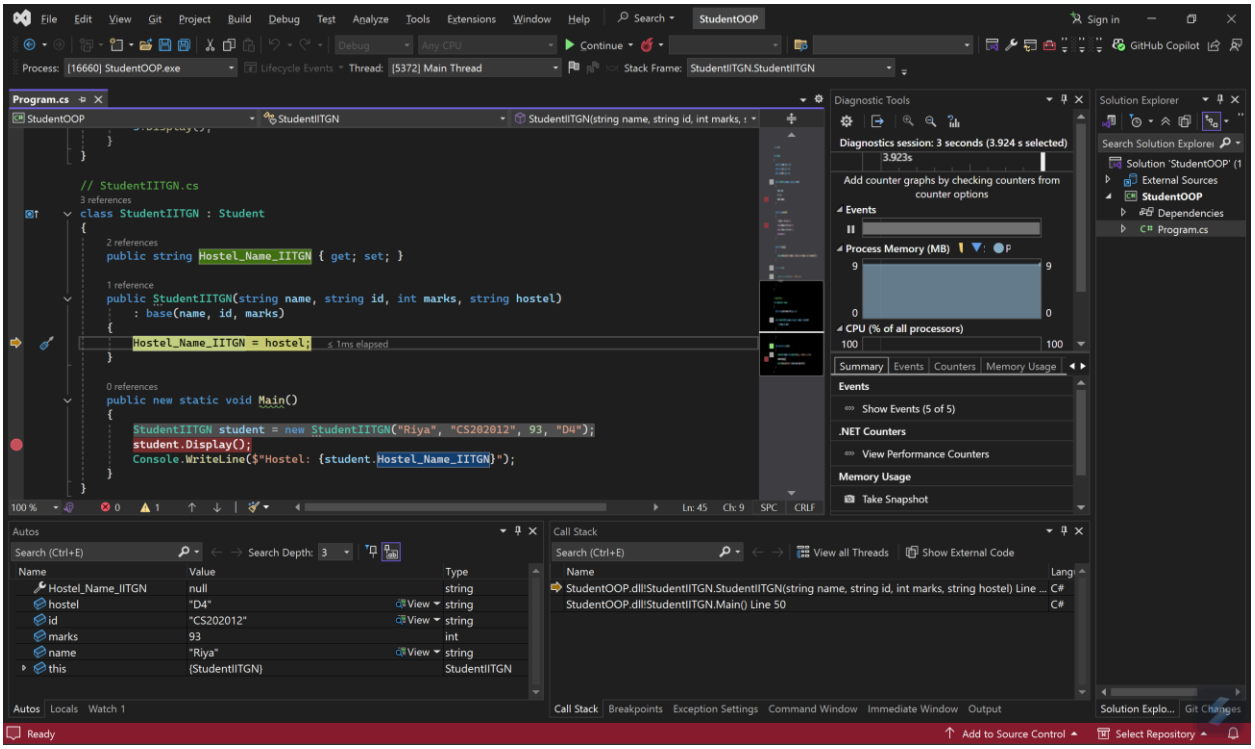
On performing step-in, the below image is obtained.

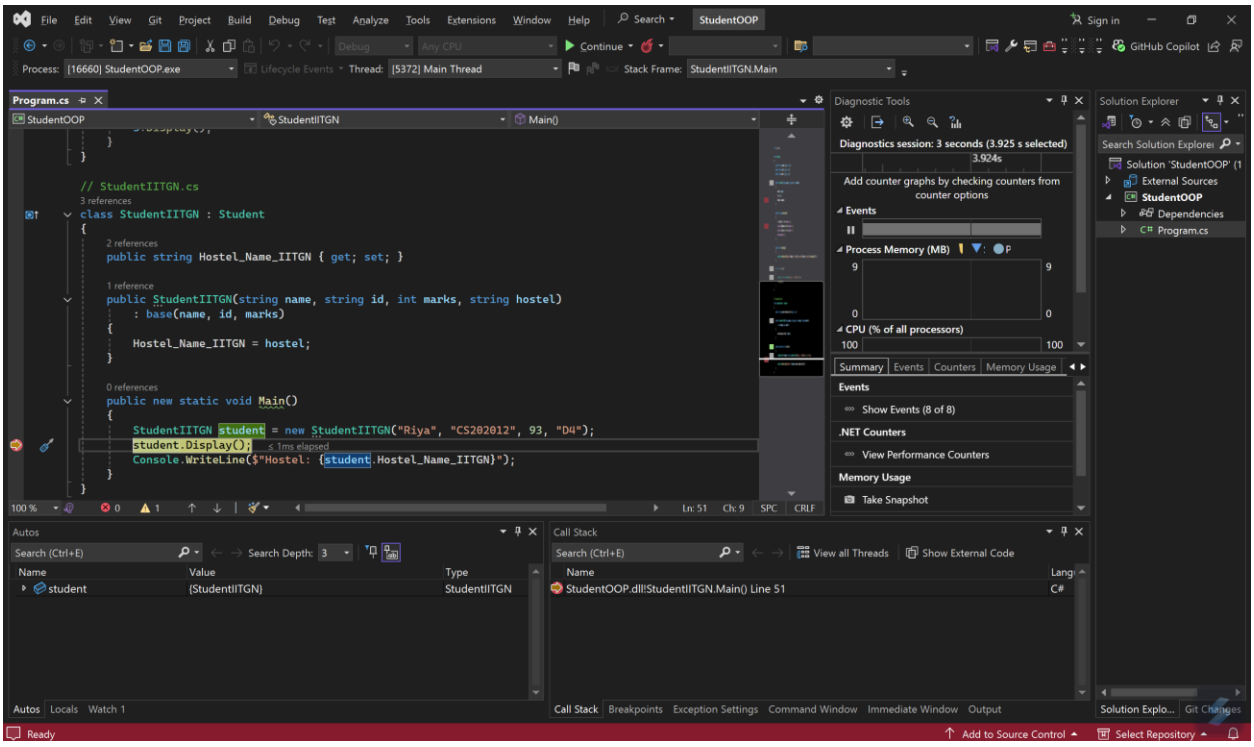


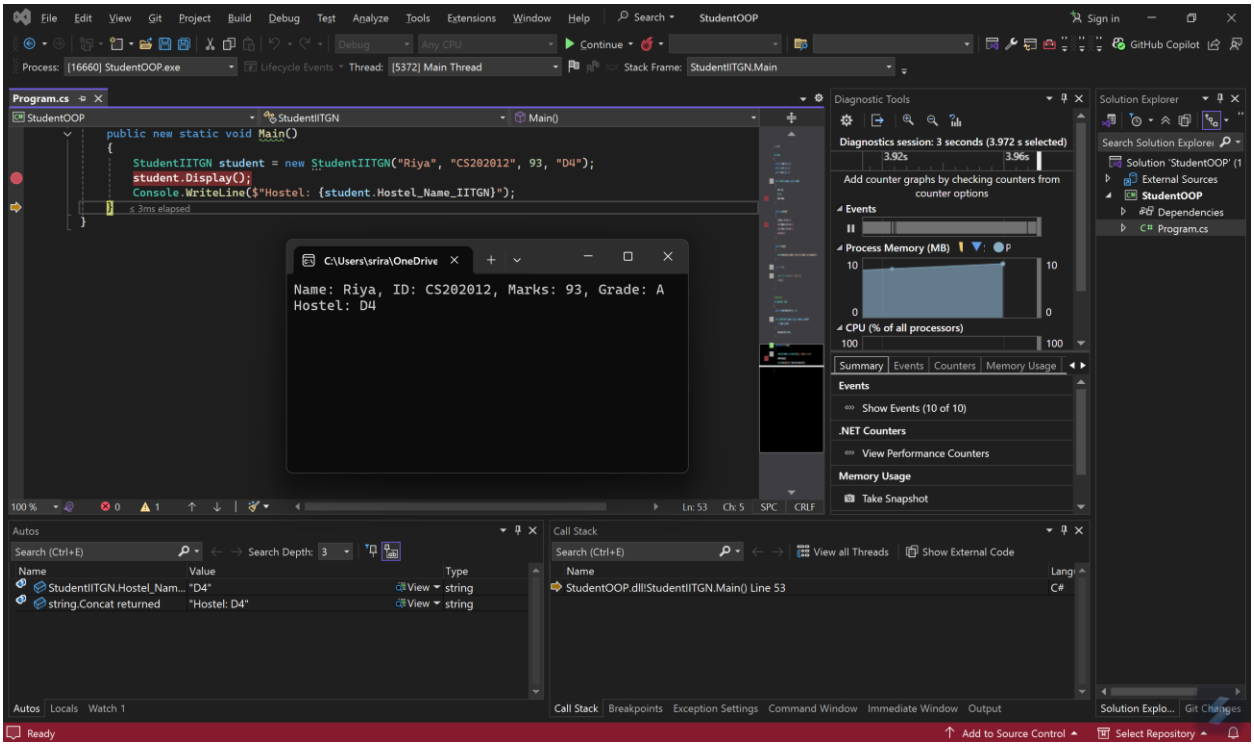

Program.cs
LoopsAndFunctions
Program
long factorial = Calcula
Console.WriteLine($"Fact
}
catch (FormatException)
{
Console.WriteLine("Inva
}
}
}
1 reference
static long CalculateFactorial(int
{
long result = 1;
for (int i = 2; i <= n; i++)
{
result *= i;
}
return result;
}
C:\Users\srira\OneDrive - iitgı
Numbers from 1 to 10:
1 2 3 4 5 6 7 8 9 10
Enter a number to find its factorial (or type 'exit' to quit): 5
Factorial of 5 is 120
Enter a number to find its factorial (or type 'exit' to quit): 6
Diagnostic Tools
Diagnostics session: 18 seconds (2 ms selected)
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
Show Events (4 of 101)
.NET Counters
View Performance Counters
Memory Usage
Take Snapshot
Solution Explorer
Solution 'LoopsAndFuncti
External Sources
LoopsAndFunctions
Dependencies
Program.cs
Autos
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
CalculateFactorial
{Method = (Int64 CalculateFactorial(Int32))}
object {System....
factorial
720
long
number
6
int
Call Stack
Search (Ctrl+E)
View all Threads
Show External Code
Name
LoopsAndFunctions.dll!Program.Main(string[] args) Line 29
C#
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready
Add to Source Control
Select Repository

Activity 4

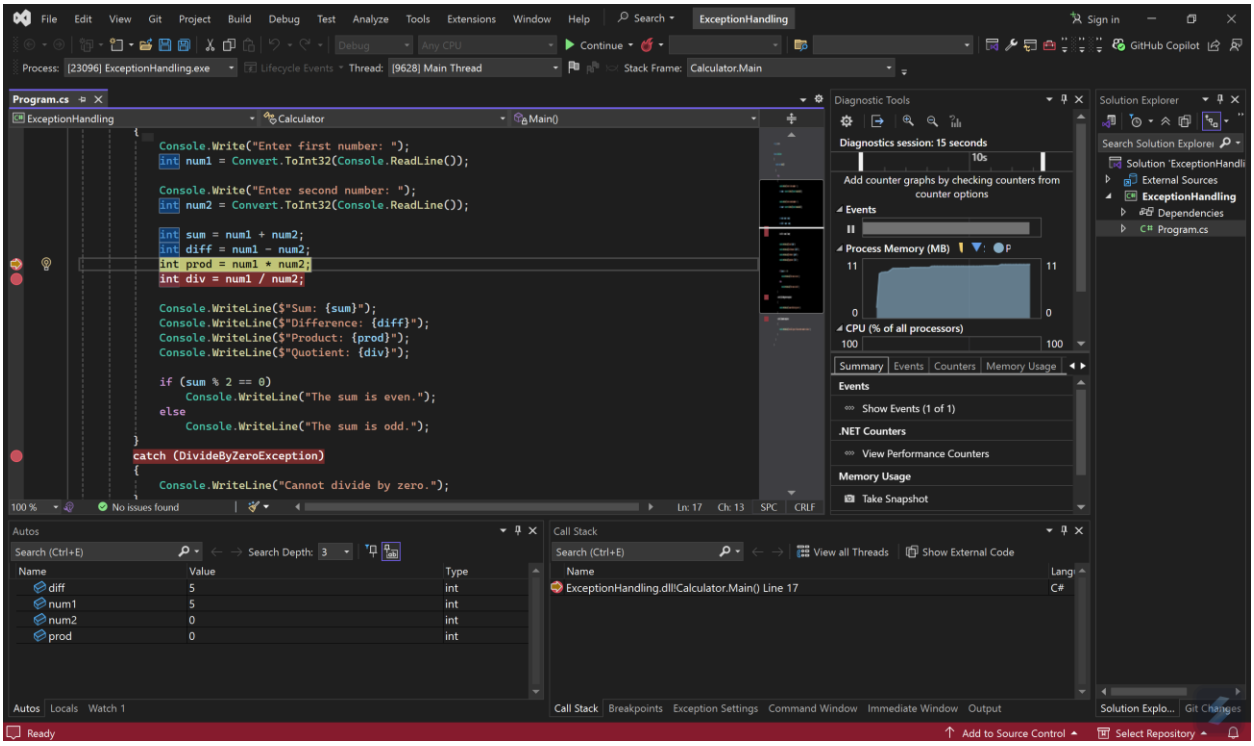

Program.cs
StudentOOP
Student
Student(string name, string id, int marks)
using System;
5 references
class Student
{
2 references
public string Name { get; set; }
2 references
public string ID { get; set; }
5 references
public int Marks { get; set; }
2 references
public Student(string name, string id, int marks)
{
Name = name;
ID = id;
Marks = marks;
}
1 reference
public string GetGrade()
{
if (Marks >= 90) return "A";
else if (Marks >= 75) return "B";
Diagnostic Tools
Diagnostics session: 3 seconds
Add counter graphs by checking counters from counter options
Events
Process Memory
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
Show Events (1 of 1)
.NET Counters
View Performance Counters
Memory Usage
Take Snapshot
Solution Explorer
Solution 'StudentOOP' (1
External Sources
StudentOOP
Dependencies
Program.cs
Autos
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
ID
"CS202012"
string
Marks
0
int
id
"CS202012"
string
marks
93
int
this
{StudentIITGN}
Student {Stude...
Call Stack
Search (Ctrl+E)
View all Threads
Show External Code
Name
StudentOOP.dll!Student.Student(string name, string id, int marks) Line 14
StudentOOP.dll!StudentIITGN.StudentIITGN(string name, string id, int marks, string hostel) Line ...
StudentOOP.dll!StudentIITGN.Main() Line 50
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready
Add to Source Control
Select Repository

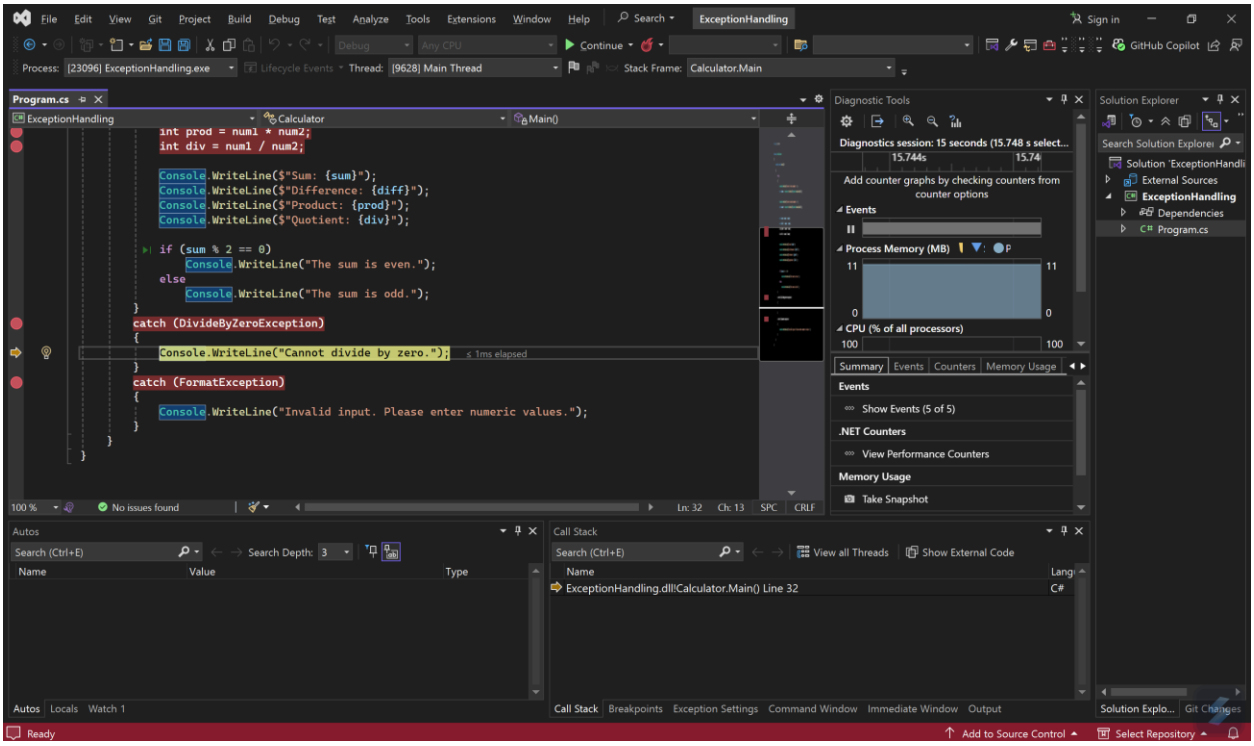

Program.cs
StudentOOP
StudentIITGN
StudentIITGN(string name, string id, int marks, s
// StudentIITGN.cs
3 references
class StudentIITGN : Student
{
2 references
public string Hostel_Name_IITGN { get; set; }
1 reference
public StudentIITGN(string name, string id, int marks, string hostel)
: base(name, id, marks)
{
Hostel_Name_IITGN = hostel;
≤ 1ms elapsed
}
0 references
public new static void Main()
{
StudentIITGN student = new StudentIITGN("Riya", "CS202012", 93, "D4");
student.Display();
Console.WriteLine($"Hostel: {student.Hostel_Name_IITGN}");
}
}
Diagnostic Tools
Diagnostics session: 3 seconds (3.924 s selected)
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
Show Events (5 of 5)
.NET Counters
View Performance Counters
Memory Usage
Take Snapshot
Solution Explorer
Solution 'StudentOOP' (1
External Sources
StudentOOP
Dependencies
Program.cs
Autos
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
Hostel_Name_IITGN
null
string
hostel
"D4"
string
id
"CS202012"
string
marks
93
int
name
"Riya"
string
this
{StudentIITGN}
StudentIITGN
Call Stack
Search (Ctrl+E)
View all Threads
Show External Code
Name
StudentOOP.dll!StudentIITGN.StudentIITGN(string name, string id, int marks, string hostel) Line ...
StudentOOP.dll!StudentIITGN.Main() Line 50
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready
Add to Source Control
Select Repository


// StudentIITGN.cs
class StudentIITGN : Student
public string Hostel_Name_IITGN { get; set; }
public StudentIITGN(string name, string id, int marks, string hostel)
: base(name, id, marks)
Hostel_Name_IITGN = hostel;
public new static void Main()
StudentIITGN student = new StudentIITGN("Riya", "CS202012", 93, "D4");
student.Display();
Console.WriteLine($"Hostel: {student.Hostel_Name_IITGN}");


Name: Riya, ID: CS202012, Marks: 93, Grade: A
Hostel: D4

Activity 5


ExceptionHandling
Program.cs
Console.Write("Enter first number: ");
int num1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter second number: ");
int num2 = Convert.ToInt32(Console.ReadLine());
int sum = num1 + num2;
int diff = num1 - num2;
int prod = num1 * num2;
int div = num1 / num2;
Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Difference: {diff}");
Console.WriteLine($"Product: {prod}");
Console.WriteLine($"Quotient: {div}");
if (sum % 2 == 0)
Console.WriteLine("The sum is even.");
else
Console.WriteLine("The sum is odd.");
catch (DivideByZeroException)
Console.WriteLine("Cannot divide by zero.");
Diagnostic Tools
Diagnostics session: 15 seconds
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
Show Events (1 of 1)
.NET Counters
View Performance Counters
Memory Usage
Take Snapshot
Autos
Name
Value
Type
diff
5
int
num1
5
int
num2
0
int
prod
0
int
Call Stack
ExceptionHandling.dll!Calculator.Main() Line 17
Solution Explorer
Ready


ExceptionHandling
Program.cs
int prod = num1 * num2;
int div = num1 / num2;
Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Difference: {diff}");
Console.WriteLine($"Product: {prod}");
Console.WriteLine($"Quotient: {div}");
if (sum % 2 == 0)
Console.WriteLine("The sum is even.");
else
Console.WriteLine("The sum is odd.");
catch (DivideByZeroException)
Console.WriteLine("Cannot divide by zero.");
catch (FormatException)
Console.WriteLine("Invalid input. Please enter numeric values.");
Diagnostic Tools
Diagnostics session: 15 seconds (15.748 s select...
Add counter graphs by checking counters from counter options
Events
Process Memory (MB)
CPU (% of all processors)
Summary
Events
Counters
Memory Usage
Show Events (5 of 5)
.NET Counters
View Performance Counters
Memory Usage
Take Snapshot
Autos
Call Stack
ExceptionHandling.dll!Calculator.Main() Line 32
Solution Explorer
Ready

Results and Analysis

After successfully setting up the .NET environment in Visual Studio 2022 and creating the required C# console applications, all the programs executed correctly and produced the expected outputs. In the initial program (Activity 1), the output simply displayed "Hello World!" which confirmed that the development environment was configured properly and that the project was running as expected.

In Activity 2, after taking two numeric inputs from the user, the program performed all the arithmetic operations — addition, subtraction, multiplication, and division — and printed the results using `Console.WriteLine()`. The condition to check whether the sum was even or odd worked well. One interesting observation here was regarding division. Initially, the program used floating-point division, and when the second number was entered as zero, instead of showing an error, it returned "Infinity." This behavior was unexpected at first, but on researching a bit, it became clear that it was due to how floating-point division works in C#. Later, this was handled using a proper `if` condition to manually check for zero before dividing.

In Activity 3, both the loops — `for` and `while` — worked as expected. The `for` loop printed numbers from 1 to 10 correctly, and the `while` loop successfully kept asking for input until "exit" was typed. The factorial function also returned accurate values, even for higher numbers. This part of the lab helped in understanding how user inputs can be controlled and handled gracefully during runtime.

By Activity 4, object-oriented programming was introduced. The class `Student` and its subclass `StudentIITGN` were created with constructors and methods. The concept of inheritance was applied properly, and it was satisfying to see how reusable and organized the code became. Grades were calculated based on marks using conditional logic, and the subclass could display additional information like hostel name. One small issue faced here was with multiple `Main()` methods — the compiler threw an error saying multiple entry points were defined. This was later resolved by removing one of the `Main()` methods and organizing the execution into a single `Main()` method inside a `Program` class.

In Activity 5, exception handling was implemented. It helped in avoiding runtime crashes due to invalid inputs and division by zero. Using `try-catch`, the program could now inform the user about the mistake instead of abruptly stopping.

Activity 6 involved debugging using Visual Studio. Breakpoints were placed strategically, and used Step Into, Step Over, and Step Out to observe how control flows during program execution. This gave a better understanding of the internal working of the program and was useful in identifying and fixing logical errors, if any.

Discussion and Conclusion

The Visual Studio interface has so many options, and configuring the project settings was new to most of me. But once the initial setup was done properly, the rest of the tasks became much easier to follow. One major challenge faced was regarding exception

handling and understanding how the compiler reacts to different types of division — especially the case with floating-point division not throwing an error. That took a bit of reading and trial-and-error to figure out.

Another minor but confusing issue was the compiler error about multiple entry points. I was not very clear on why the program was throwing that error, but after checking the class structure and method names, I understood that C# only allows one `Main()` method as the entry point in a console application. This was a good learning experience because it taught us about project structure and execution flow in a real-world setting.

The lab also reinforced the importance of good naming conventions and file organization. Initially, naming was done randomly, but later it became clear how clean naming helps in managing multiple classes and understanding the code better. Debugging with breakpoints also gave a new perspective, especially for tracing logic in loops and function calls.

In conclusion, this lab gave a very practical exposure to building C# console applications and understanding the .NET environment. From basic syntax and control structures to object-oriented programming and exception handling, everything was covered step by step.