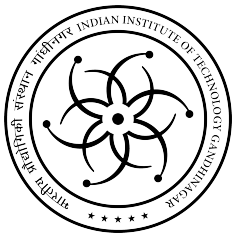


Assignment 3

CS 202: Software Tools And Techniques for CSE

Shubham Agrawal
22110249
Department of Computer Science and Engineering

Supervisor:
Prof. Shouvick Mondal

Indian Institute of Technology Gandhinagar

Thursday 10th April, 2025

Contents

1 Introduction 1

2 LAB-9 *dated 20/03/25* 2

2.1 Lab Topic 2

2.2 Overview 2

2.3 Objectives 2

2.4 Set-up and Tools 2

2.5 Methodology and Execution 3

2.6 Results and Analysis 7

2.7 Discussion and Conclusion 15

2.8 Files and Code Location 16

3 LAB-10 *dated 27/03/25* 17

3.1 Lab Topic 17

3.2 Overview 17

3.3 Objectives 17

3.4 Set-up and Tools 17

3.5 Methodology and Execution 18

3.6 Results 20

3.7 Discussion and Conclusion 30

3.8 Files and Code Location 31

1 Introduction

This assignment contains a detailed and structured report for the Software Tools and Techniques for CSE (CS202) course labs carried out in March 2025. There were a total of two labs which were performed on 20th, and 27th March.

2 LAB-9 *dated 20/03/25*

2.1 Lab Topic

Module Dependency and Cohesion Analysis

2.2 Overview

In this lab, I worked with two real-world codebases—one written in Python and another in Java—to study their structural properties. With pydeps, I inspected the internal dependencies of the Python project and detected problematic modules with high coupling or dead links. I then used the LCOM.jar tool to calculate different cohesion measures for Java classes, which helped me evaluate their internal design consistency. From the results, I investigated possibilities for modularization and refactoring, aiming to improve the overall architecture and maintainability of the projects.

2.3 Objectives

The objective of this lab was to perform an in-depth analysis of software dependencies and class cohesion using automated tools. I aimed to generate and interpret Python module dependency graphs using pydeps, and evaluate Java class cohesion using LCOM.jar.

2.4 Set-up and Tools

The following tools and set-ups were used to complete the lab objectives:

- **Operating System:** Windows 11
- **Python 3.10:** Coding Language Used.
- **Java JDK 17:** Coding Language Used.
- **Lightening AI:** Remote development platform with GPU availability along with an IDE similar to VS-code.
- **Tools:** `pydeps` and `LCOM`

- **SEART GitHub Search Engine:** Used for selecting repository based on specific criteria.

2.5 Methodology and Execution

2.5.1 pydeps Set-up

pydeps was installed along with its other dependencies, mainly **graphviz**, using:

```
sudo apt install graphviz
pip install pydeps
```

2.5.2 Selecting 1 Python Repository

- Used SEART Github Search Engine for selecting 3 large scale open source repository of a real-world project.
- Defined Selection criteria as:
    1. **No. of Commits-** Min 1000
    2. **No. of Stars-** Min 500
    3. **No. of Forks-** Min 500
    4. **No. of Branches-** Min 3
    5. **Size of Code Base-** Min 500 lines
    6. **Language-** Python
- Defined the above selection criteria to ensure that I find one repository that is a real-life project, is workable with, has decent number of commits, as well as it should be a python project and finally, aligns with the lab objective.
- Finally, out of the results, I picked one at random:
    1. Apscheduler `https://github.com/agronholm/apscheduler.git`

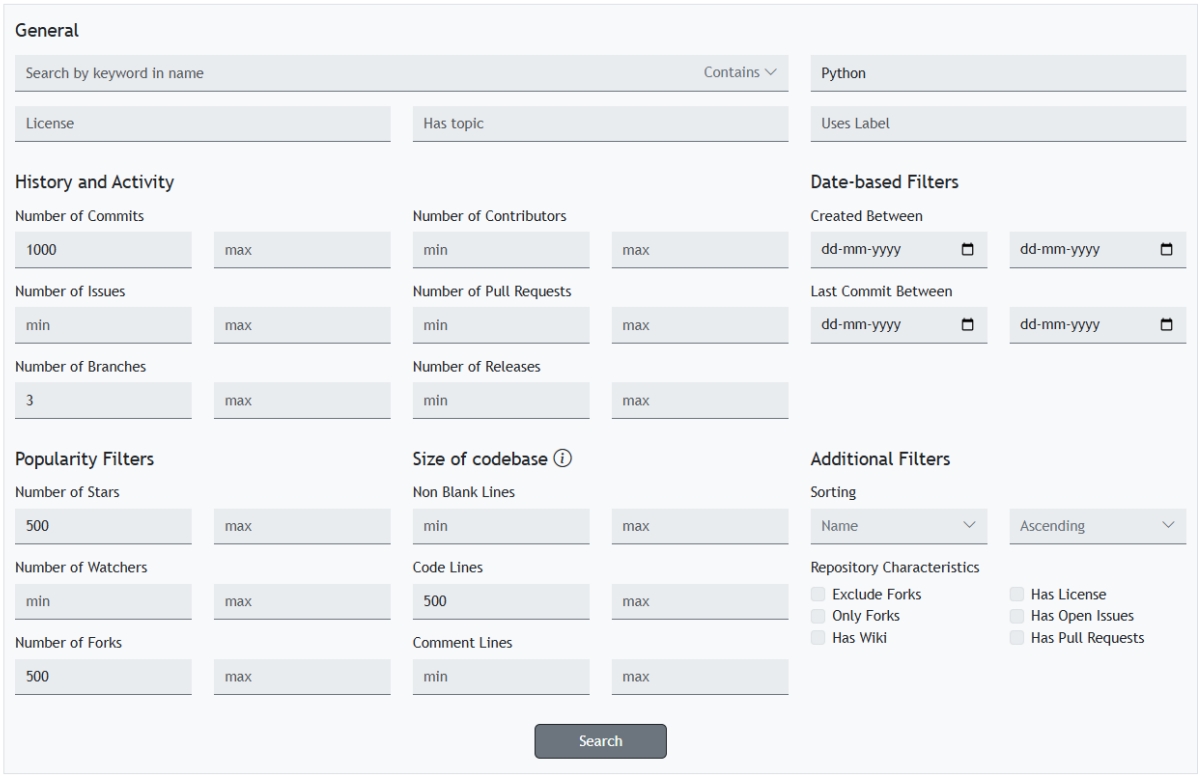


Figure 1: SEART Github Search Parameters

2.5.3 Cloning the python repository

`agronholm/apscheduler` was cloned using:

```
git clone https://github.com/agronholm/apscheduler.git
```

2.5.4 Running Pydeps

After cloning the repository, I ran pydeps on it using the following commands:

```
cd apscheduler
cd src
pydeps apscheduler
pydeps apscheduler --show-deps > dependencies.json
```

2.5.5 Pydeps Result Analysis

I wrote an .ipynb file called `analysis.ipynb` to run analysis on the `dependencies.json` file to analyse the following:

1. Fan-in and Fan-out of each module
2. Highly Coupled Modules
3. Cyclic dependencies
4. Unused and disconnected Modules
5. Depth of Dependencies
6. Core Modules and their impacts

Note: `analysis.ipynb` file was too large to be displayed here. Kindly access it from the github repo link attached at the end of the report.

2.5.6 Java Set-up

Java was installed along with its other dependencies, mainly **tree**, using:

```
sudo apt install openjdk-17-jdk
sudo apt install tree
```

2.5.7 Selecting 1 Java Repository

- Used SEART Github Search Engine for selecting 3 large scale open source repository of a real-world project.
- Defined Selection criteria as:
    1. **No. of Commits-** Min 1000
    2. **No. of Stars-** Min 500
    3. **No. of Forks-** Min 500
    4. **No. of Branches-** Min 3
    5. **Size of Code Base-** Min 500 lines
    6. **Language-** Java

- Defined the above selection criteria to ensure that I find one repository that is a real-life project, is workable with, has decent number of commits, as well as it should be a java project and finally, aligns with the lab objective.
- Finally, out of the results, I picked one at random:
    1. Java-WebSocket `https://github.com/TooTallNate/Java-WebSocket.git`

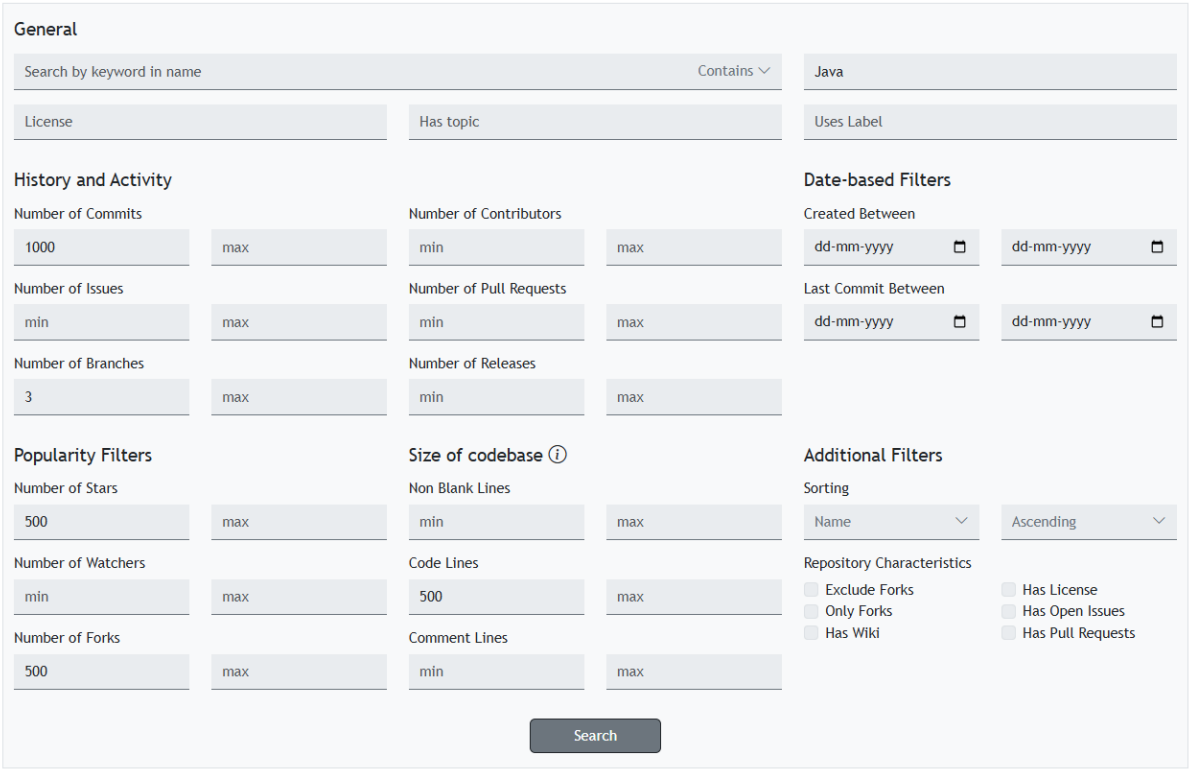


Figure 2: SEART Github Search Parameters

2.5.8 Cloning the java repository

`TooTallNate/Java-WebSocket` was cloned using:

```
git clone https://github.com/TooTallNate/Java-WebSocket.git
```

2.5.9 Running LCOM

After cloning the repository, I downloaded the LCOM file provided in the lab, and ran it on the repo using the following commands:

```
cd Java-WebSocket
java -jar LCOM.jar -i src/main/java/org/java_websocket -o lcom_output/
```

2.5.10 LCOM Result Analysis

I wrote an .ipynb file called `java_analysis.ipynb` to run analysis on the lcom output file to analyse the following:

1. Full LCOM metric table
2. Classes with High LCOM Values

Note: `java_analysis.ipynb` file was too large to be displayed here. Kindly access it from the github repo link attached at the end of the report.

2.6 Results and Analysis

2.6.1 Outputs

Python Repository Selection: I shortlisted repositories based on criteria such as the number of commits, GitHub stars, and forks to ensure real-life large scale projects. Using the SEART Github Search Engine, I finally shortlisted the following repositories:

1. Apscheduler `https://github.com/agronholm/apscheduler.git`

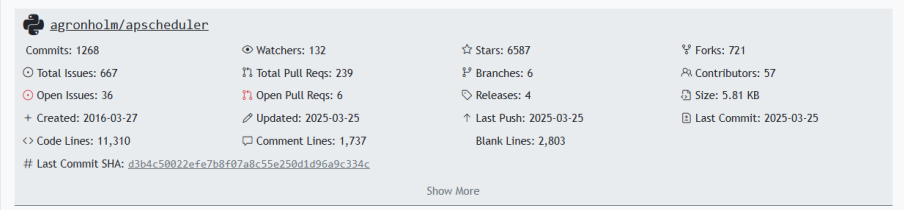


Figure 3: Apscheduler

Pydeps Results: The dependencies graph obtained as a result of running pydeps was large and hence cannot be fully displayed here. A screenshot of a small part of graph is attached. For the full graph, refer to the github repo link.

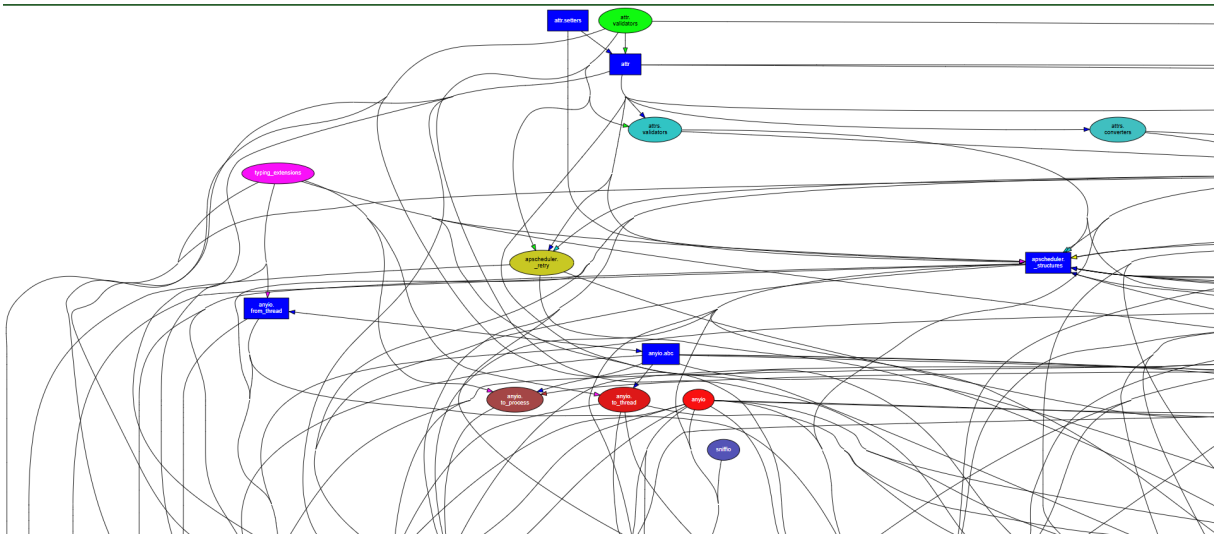


Figure 4: Dependencies Graph

1. Fan-in and Fan-out of each module

	module	fan_in	fan_out	total_coupling
	__main__	0	45	45
	attrs	30	4	34
	apscheduler._schedulers.async_	4	26	30
	apscheduler.abc	21	4	25
	apscheduler._structures	15	10	25
	apscheduler._utils	21	3	24
	attr	19	3	22
	apscheduler	10	11	21
	apscheduler._events	14	6	20
	apscheduler.eventbrokers.base	6	11	17
	attr.validators	15	0	15
	apscheduler.datastores.sqlalchemy	1	13	14
	apscheduler._schedulers.sync	3	11	14
	apscheduler.datastores.mongodb	1	12	13
	anyio	13	0	13
	apscheduler.eventbrokers.psycopg	1	12	13
	apscheduler.eventbrokers.asyncpg	1	11	12
	apscheduler._exceptions	12	0	12
	apscheduler._converters	11	0	11
	apscheduler._decorators	3	7	10
	apscheduler.datastores.memory	2	8	10
	apscheduler._enums	10	0	10
	apscheduler.eventbrokers.mqtt	1	9	10
	anyio.abc	8	1	9
...				
	apscheduler.eventbrokers	2	0	2
	apscheduler.executors	2	0	2
	apscheduler.triggers	1	0	1
	sniffio	1	0	1

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Figure 5: Fan In Fan out

2. Highly Coupled Modules

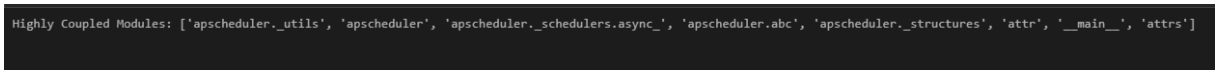


Figure 6: Highly Coupled Modules

3. Cyclic dependencies

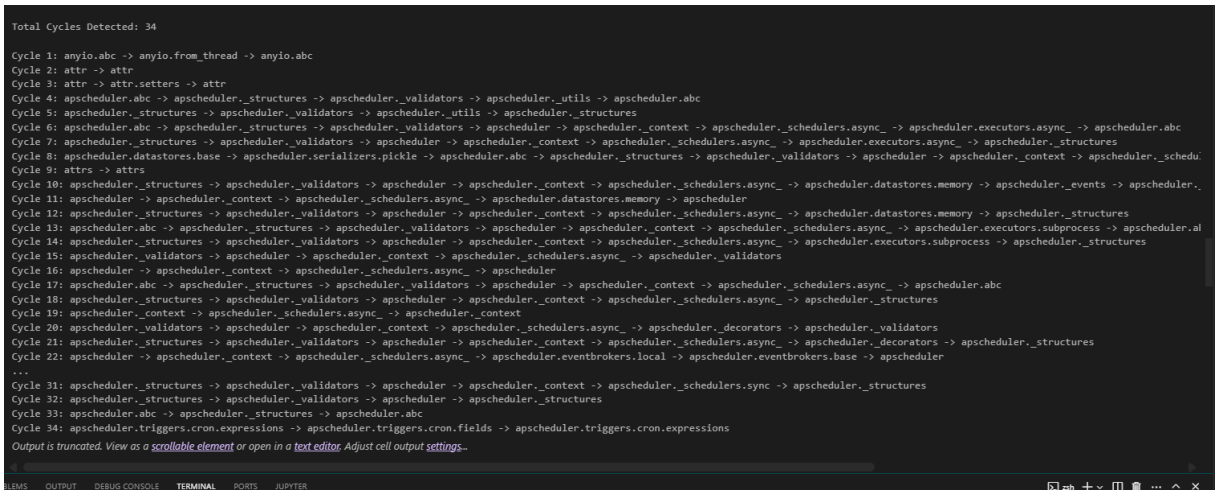


Figure 7: Cyclic Dependencies

4. Unused and disconnected Modules

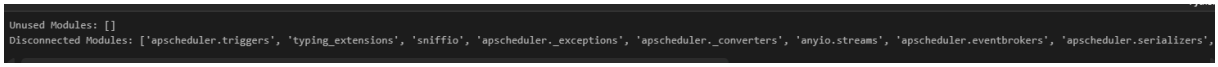


Figure 8: Unused and Disconnected Modules

5. Depth of Dependencies

Figure 9: Depths of Dependencies

6. Core Modules and their impacts

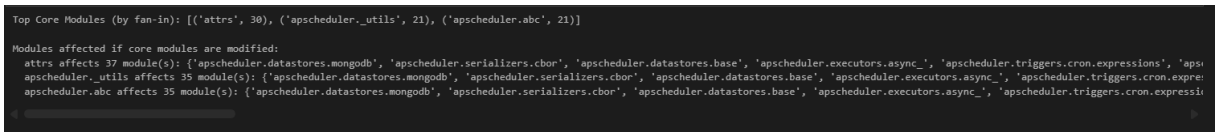


Figure 10: Core Modules

Java Repository Selection: I shortlisted repositories based on criteria such as the number of commits, GitHub stars, and forks to ensure real-life large scale projects. Using the SEART Github Search Engine, I finally shortlisted the following repositories:

1. Java-WebSocket `https://github.com/TooTallNate/Java-WebSocket.git`

Figure 11: Java-WebSocket

LCOM Results:

1. Full LCOM Metric Table

Full LCOM Metrics Table:

Project Name	Package Name	Type Name	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
java_websocket	org.java_websocket.server	SSLParametersWebSocketServerFactory	0.0	0.0	1.0	1.0	-0.000000	0.000000
java_websocket	org.java_websocket.server	DefaultWebSocketServerFactory	6.0	0.0	4.0	4.0	0.000000	1.000000
java_websocket	org.java_websocket.server	DefaultSSLWebSocketServerFactory	12.0	9.0	4.0	4.0	0.700000	0.666667
java_websocket	org.java_websocket.server	CustomSSLWebSocketServerFactory	0.0	0.0	1.0	1.0	-0.000000	0.000000
java_websocket	org.java_websocket.server	WebSocketServer	2384.0	2212.0	33.0	9.0	0.946976	0.097222
java_websocket	org.java_websocket.server	WebSocketWorker	7.0	4.0	3.0	2.0	0.500000	0.400000
java_websocket	org.java_websocket	SSLSocketChannel2	244.0	110.0	4.0	1.0	0.867725	0.000000
java_websocket	org.java_websocket	WrappedByteChannel	10.0	0.0	5.0	5.0	0.000000	-1.000000
java_websocket	org.java_websocket	WebSocket	325.0	0.0	26.0	26.0	0.000000	-1.000000
java_websocket	org.java_websocket	AbstractWrappedByteChannel	0.0	0.0	1.0	1.0	-0.000000	0.000000
java_websocket	org.java_websocket	WebSocketImpl	1319.0	1098.0	18.0	6.0	0.942292	0.107143
java_websocket	org.java_websocket	WebSocketAdapter	15.0	15.0	6.0	6.0	1.000000	1.000000
java_websocket	org.java_websocket	WebSocketFactory	1.0	0.0	2.0	2.0	0.000000	-1.000000
java_websocket	org.java_websocket	WebSocketServerFactory	6.0	0.0	4.0	4.0	0.000000	-1.000000
java_websocket	org.java_websocket	AbstractWebSocket	110.0	84.0	5.0	4.0	0.880208	0.235294
java_websocket	org.java_websocket	SSLSocketChannel	94.0	35.0	6.0	3.0	0.816176	0.166667
java_websocket	org.java_websocket	SocketChannelIOHelper	6.0	0.0	4.0	4.0	0.000000	1.000000
java_websocket	org.java_websocket	WebSocketListener	120.0	0.0	16.0	16.0	0.000000	-1.000000
java_websocket	org.java_websocket.util	ByteBufferUtils	3.0	0.0	3.0	3.0	0.000000	1.000000
java_websocket	org.java_websocket.util	Charsetfunctions	36.0	0.0	9.0	7.0	1.000000	0.777778
java_websocket	org.java_websocket.util	Base64	55.0	5.0	4.0	3.0	0.912698	0.133333
java_websocket	org.java_websocket.util	OutputStream	7.0	0.0	2.0	2.0	0.660000	0.333333
java_websocket	org.java_websocket.util	NamedThreadFactory	0.0	0.0	1.0	1.0	0.500000	0.000000
...								
java_websocket	org.java_websocket.extensions	IExtension	45.0	0.0	10.0	10.0	0.000000	-1.000000
java_websocket	org.java_websocket.extensions.permessage_deflate	PerMessageDeflateExtension	168.0	146.0	7.0	5.0	0.944079	0.200000
java_websocket	org.java_websocket.protocols	Protocol	11.0	1.0	3.0	1.0	0.777778	0.000000
java_websocket	org.java_websocket.protocols	IProtocol	6.0	0.0	4.0	4.0	0.000000	-1.000000

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Figure 12: LCOM Result Table

2. Classes with High LCOM Values

Classes with High LCOM (Low Cohesion, Consider Refactoring):

Project Name	Package Name	Type Name	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
java_websocket	org.java_websocket.server	WebSocketServer	2384.0	2212.0	33.0	9.0	0.946976	0.097222
java_websocket	org.java_websocket	SSLSocketChannel2	244.0	110.0	4.0	1.0	0.867725	0.000000
java_websocket	org.java_websocket	WebSocket	325.0	0.0	26.0	26.0	0.000000	-1.000000
java_websocket	org.java_websocket	WebSocketImpl	1319.0	1098.0	18.0	6.0	0.942292	0.107143
java_websocket	org.java_websocket	AbstractWebSocket	110.0	84.0	5.0	4.0	0.880208	0.235294
java_websocket	org.java_websocket	WebSocketListener	120.0	0.0	16.0	16.0	0.000000	-1.000000
java_websocket	org.java_websocket.drafts	Draft_6455	1316.0	1147.0	22.0	9.0	0.944444	0.163636
java_websocket	org.java_websocket.drafts	Draft	400.0	394.0	26.0	21.0	0.946429	0.724138
java_websocket	org.java_websocket.framing	FramedataImpl1	129.0	68.0	3.0	3.0	0.774436	0.150000
java_websocket	org.java_websocket.client	WebSocketClient	2400.0	1797.0	22.0	10.0	0.937662	0.128205
java_websocket	org.java_websocket.extensions.permessage_deflate	PerMessageDeflateExtension	168.0	146.0	7.0	5.0	0.944079	0.200000

+ Code + Markdown

Figure 13: High LCOM Value Classes

2.6.2 Observations

Pydeps:

1. **Fan-in and Fan-out of each module:** The module `__main__` stood out with an exceptionally high fan-out of 45, meaning it imports a significant number of other modules. On the other hand, the module `attrs` had the highest fan-in value of 30, indicating it is widely reused across the system.

2. **Highly Coupled Modules:** The module with the highest total coupling (fan-in + fan-out) was `__main__` with a coupling score of 45, followed by `attrs` (34) and `apscheduler._schedulers.async_` (30).

3. **Cyclic dependencies:** Cyclic dependencies were detected in modules such as `apscheduler._schedulers.async_` and related scheduler/executor components. Cyclic dependencies reduce maintainability by tightly coupling modules, making it difficult to modify one without affecting others. They complicate testing, as modules can't be easily isolated, and increase the risk of introducing bugs during refactoring. This interdependence also hampers readability and makes the overall system harder to understand, evolve, and debug.

4. **Unused and disconnected Modules:** The analysis revealed several disconnected modules—those neither importing nor being imported by any other module. These may represent dead code, placeholders, or utilities not yet integrated into the system. While not inherently problematic, disconnected modules can inflate the codebase unnecessarily and should be reviewed for relevance. If confirmed as unused, they can be removed to simplify maintenance and improve overall clarity.

5. **Depth of Dependencies:** The depth analysis revealed that modules like `apscheduler._schedulers.sync` and `apscheduler._schedulers.async_` were positioned deeper in the graph, implying more transitive dependencies. This increases the risk of cascading changes and can negatively impact maintainability.

6. **Core Modules:** Based on high fan-in values, the top three core modules identified were:

   (a) attrs (fan-in: 30)

   (b) apscheduler.abc (fan-in: 21)

(c) apscheduler._utils (fan-in: 21)

These modules are foundational and serve as common dependencies across many parts of the system. Any changes to them could have widespread ripple effects.

7. **Impact Propogation:** Using reverse dependency analysis, I found that if `attrs` were modified, at least 30 modules could be affected. Similarly, modifying `apscheduler.abc` or `apscheduler._utils` would impact 21 modules each. These insights emphasize the importance of regression testing and backward compatibility when updating core modules.

LCOM:

The LCOM (Lack of Cohesion of Methods) analysis highlighted varying degrees of cohesion across different classes in the java_websocket project. Classes such as WebSocketServer, SSLSocketChannel2, and WebSocketImpl exhibited high LCOM1 and LCOM2 values, indicating a low level of cohesion, meaning their methods interact with relatively few shared attributes. This may suggest that these classes are doing too many unrelated things and could benefit from refactoring or decomposition into smaller, more focused classes.

On the other hand, classes like SSLParametersWebSocketServerFactory, CustomSSLWebSocketServerFactory, and AbstractWrappedByteChannel showed very low or zero LCOM scores, implying high cohesion and strong internal consistency—where most methods operate on common attributes. These classes are likely well-structured and follow the Single Responsibility Principle.

Additionally, negative or extremely low LCOM5 and YALCOM values in certain classes such as WebSocket and WrappedByteChannel indicate strong method-attribute coupling, potentially making them easier to maintain and understand. In contrast, the classes with high YALCOM values (close to 1) might contain methods with little interdependence, which could lead to maintenance challenges if not carefully documented or modularized.

1. **Classes with High LCOM Values:** Several classes in the java_websocket project demonstrated notably high LCOM values, including:

   (a) WebSocketServer (LCOM1: 2384.0, LCOM2: 2212.0)

   (b) WebSocketImpl (LCOM1: 1319.0, LCOM2: 1098.0)

(c) SSLSocketChannel2 (LCOM1: 244.0, LCOM2: 110.0)

A high LCOM value generally indicates that the class contains methods that do not interact with common fields, implying poor cohesion. This suggests that the class is performing multiple unrelated tasks, violating the Single Responsibility Principle. Such design can lead to code that is harder to maintain, test, and understand.

In this repo, classes like WebSocketServer and WebSocketImpl are prime candidates for functional decomposition. These classes likely encapsulate diverse functionalities that could be split into smaller, more cohesive helper classes or services, each handling a single responsibility.

2.6.3 Key Insights

Through this lab, I gained a deeper understanding of software modularity and design quality by analyzing both Python and Java codebases. Using pydeps, I was able to construct and interpret module-level dependency graphs. This helped me identify core modules with high fan-out, detect cyclic dependencies that could hinder maintainability, and spot unused or disconnected modules. These findings offered valuable clues about the overall structure and potential architectural flaws in the project. In the Java codebase, I computed various LCOM metrics to measure class cohesion. High LCOM values pointed to poor cohesion and hinted at the need for functional decomposition, while low values confirmed well-encapsulated responsibilities. This analysis allowed me to objectively assess the internal design of classes and make informed decisions about refactoring.

2.6.4 Comparisons

The pydeps and LCOM analyses complemented each other by offering different perspectives on software quality. While pydeps focused on inter-module dependencies, helping me understand how different components interact, the LCOM metrics provided insights into intra-class cohesion, revealing how well methods within a class work together. Pydeps was useful for identifying architectural issues like dependency cycles and critical modules, whereas LCOM highlighted design concerns within individual classes. Together, these tools enabled a more holistic evaluation of the code, allowing me to address both system-level structure and internal class

design. This dual approach helped me better appreciate the importance of both modularity and cohesion in writing maintainable, scalable software.

2.7 Discussion and Conclusion

2.7.1 Challenges Faced

During the lab, one of the main challenges I faced was navigating and parsing large, nested JSON structures for module dependency analysis.

2.7.2 Reflection

This lab allowed me to reflect on how code quality goes beyond functional correctness. I realized the importance of structure, modularity, and cohesion in maintaining and scaling software projects. Working through both Python and Java ecosystems reinforced the idea that while the syntax and tools may differ, the core principles of clean, maintainable design remain consistent. It also made me appreciate the power of static analysis in identifying subtle issues before they escalate.

2.7.3 Lessons Learned

I learned how to effectively apply static analysis tools like pydeps and LCOM calculators to extract meaningful insights from a codebase. I understood how fan-in and fan-out metrics help identify key modules and how cycles can compromise code health. From the LCOM analysis, I learned to evaluate class cohesion and identify classes that could benefit from decomposition. This lab also improved my ability to visualize and interpret metric-driven data to make design decisions.

2.7.4 Summary

Overall, this lab deepened my understanding of software design metrics and dependency management. I explored tools and techniques to assess and visualize both module-level and class-level qualities. By analyzing core modules, cycles, and cohesion values, I developed a structured approach to evaluating maintainability. The combination of hands-on scripting, metric

analysis, and reflection made this lab a valuable experience in strengthening my software engineering perspective.

2.8 Files and Code Location

All files and code related to this lab can be found in the following Github repo link:

`https://github.com/shubham-agrawal04/STT-Assignment-3`

3 LAB-10 *dated 27/03/25*

3.1 Lab Topic

Development of C# Console Applications

3.2 Overview

During the lab, I set up the .NET development environment in Visual Studio and created multiple C# console applications. I wrote programs that performed arithmetic operations, implemented control flow using if-else statements, and utilized loops and functions effectively. I also created classes and subclasses to apply object-oriented principles such as encapsulation and inheritance. Furthermore, I handled exceptions using try-catch blocks and explored Visual Studio's debugging tools to trace and correct errors during runtime.

3.3 Objectives

In this lab, my objective was to gain hands-on experience in developing C# console applications using Visual Studio. I focused on learning the basic syntax of C#, understanding control structures like loops and conditionals, and exploring object-oriented programming concepts. I aimed to build simple programs that incorporated user input, arithmetic operations, and class-based design to deepen my understanding of .NET development fundamentals.

3.4 Set-up and Tools

The following tools and set-ups were used to complete the lab objectives:

- **Operating System:** Windows 11
- **C#:** Coding language used for developing console applications.
- **Visual Studio:** Visual Studio 2022 (Community Edition) was used as the integrated development environment (IDE) for writing, running, and debugging C# programs. It provided essential features such as IntelliSense, breakpoints, and a built-in terminal for seamless development.

- **.NET SDK:** .NET 6 SDK was used as the target framework for building and executing the C# applications.

3.5 Methodology and Execution

3.5.1 Setting Up .NET Development Environment:

I started by installing Visual Studio 2022 (Community Edition) along with the latest .NET 6 SDK on my Windows 11 system. I then launched Visual Studio and created a new C# Console Application project. I verified that the project was targeting the .NET 6 framework.

3.5.2 Basic Program:

I wrote a simple welcoming program to ensure that the environment was configured correctly and functioning as expected.

3.5.3 Basic Syntax and Control Structures:

I wrote a program that accepted two numbers as input from the user and performed basic arithmetic operations—addition, subtraction, multiplication, and division. I implemented an `if-else` condition to check whether the sum of the two numbers was even or odd and displayed the results using `Console.WriteLine()` statements.

3.5.4 Implementing Loops and Functions:

In this part, I developed a program that utilized both `for` and `while` loops. The `for` loop printed numbers from 1 to 10, and the `while` loop continuously accepted user input until the word "exit" was entered. I also created a function that calculated the factorial of a number provided by the user, and then invoked this function from the main program.

3.5.5 Performing Object-Oriented Programming in C#:

To apply object-oriented programming concepts, I created a class named `Student` with properties like `Name`, `ID`, and `Marks`. I implemented a constructor to initialize these values and a method `getGrade()` to return grades based on marks. I then extended this class by creating a subclass `StudentIITGN`, adding a new property called `Hostel_Name_IITGN`. I tested

both classes by creating objects and calling their respective `Main()` methods to display student information.

3.5.6 Exception Handling:

I revisited the control structures program and modified it to include exception handling. I used `try-catch` blocks to gracefully handle division-by-zero errors and invalid input types. This ensured that the program could continue running without crashing when faced with runtime exceptions, enhancing its robustness and user-friendliness.

3.5.7 Debugging using Visual Studio Debugger:

To understand program flow and logic errors, I used the Visual Studio Debugger to step through my code. I inserted breakpoints at key points in the arithmetic, loop, and object-oriented programs. Using the debugger, I performed step-in, step-over, and step-out operations to trace how variables changed and how control moved through the code.

3.5.8 Code Structure

In order to keep the code organized and facilitate modular execution, I organized my project into several C# files corresponding to individual activities in the lab. The main file, `Program.cs`, was the entry point of the program. Here, I included a basic menu-based interface to ask the user which program to execute, from simple greetings to higher-level tasks such as object-oriented programming and exception handling.

Based on the user's input, `Program.cs` called the appropriate method from the corresponding source file. I created separate class files such as `basic.cs`, `control.cs`, `loop_Func.cs`, `oops.cs`, and `error_handling.cs`, each encapsulating logic relevant to its topic. This modular approach helped maintain clean separation of concerns, made debugging easier, and allowed me to reuse and test each part independently. Each file had static methods that could be directly invoked from the main menu, ensuring that all functionalities were accessible from a single consolidated interface.

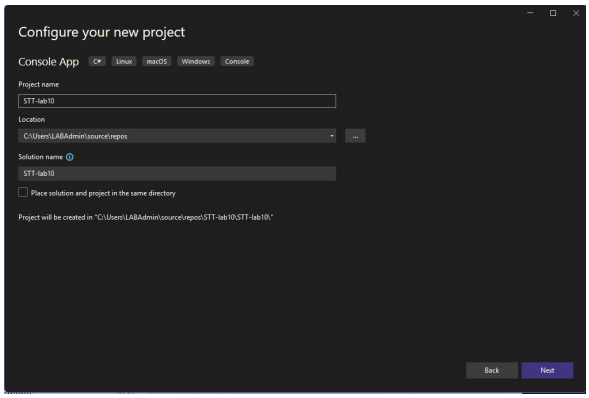
Note: The codes for the following program couldn.t be included in the report due to their large size. Kindly refer to them from the github repo link present at the end of the report.

3.6 Results

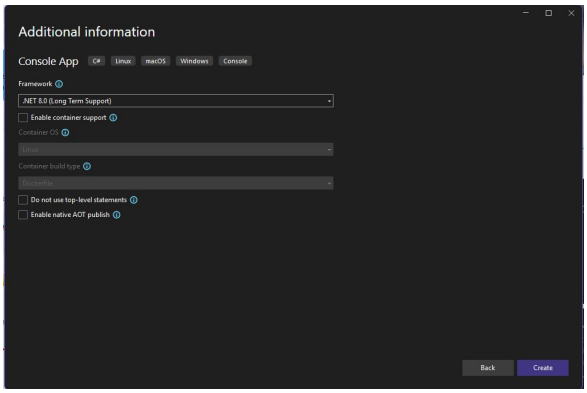
3.6.1 Outputs

Following are the screenshots of each activity carried out:

Setting Up .NET Development Environment:



(a) Creating A New Project



(b) Setting up .NET Environment

Figure 14: Initial Project Setup in Visual Studio

Basic Program:

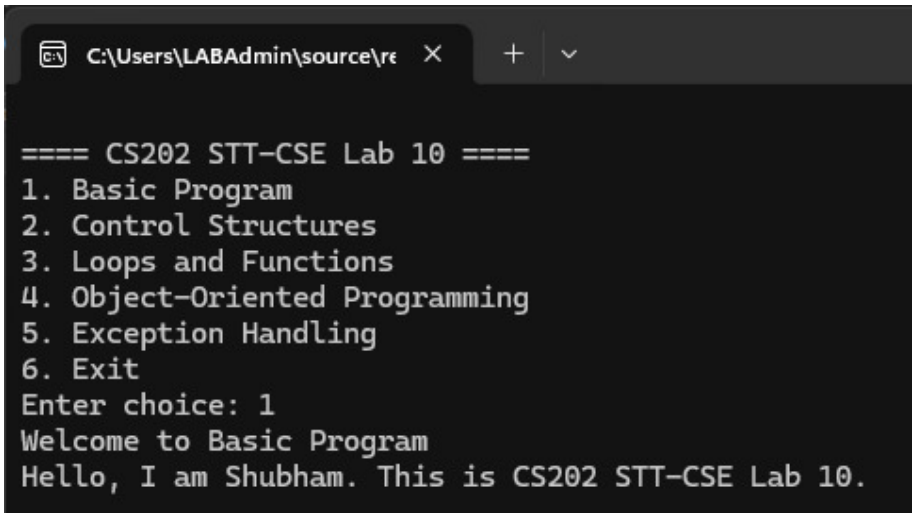


Figure 15: Basic Program

Basic Syntax and Control Structures:

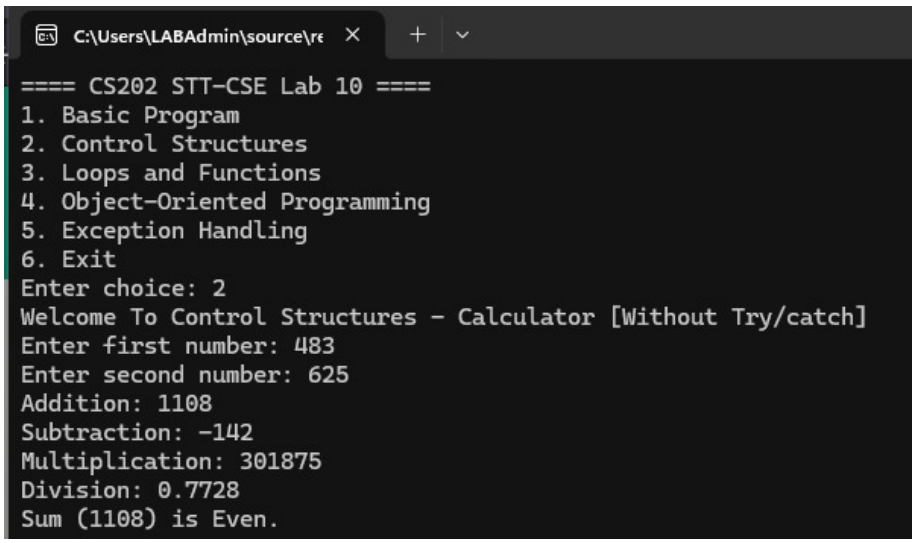


Figure 16: Control Structures

.

Implementing Loops and Functions:

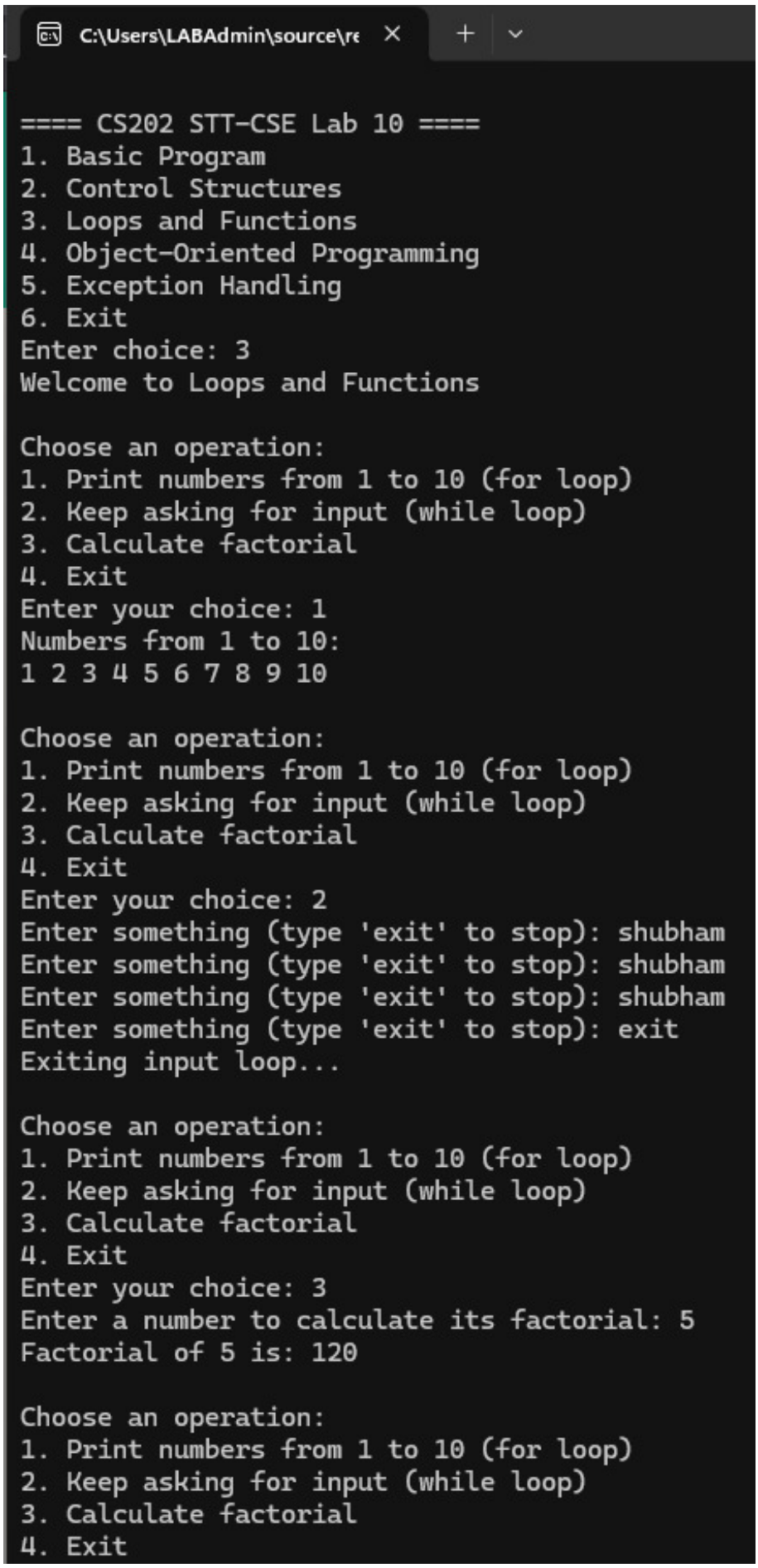


Figure 17: Loops and Functions

Performing Object-Oriented Programming in C#:

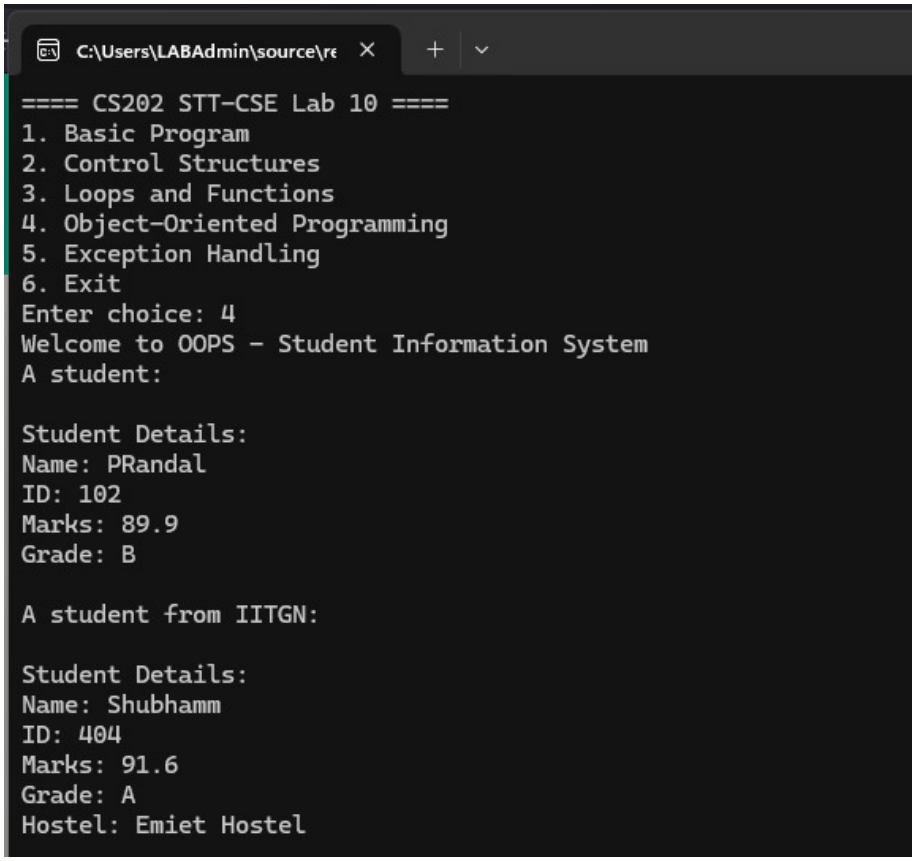


Figure 18: OOP

Exception Handling:

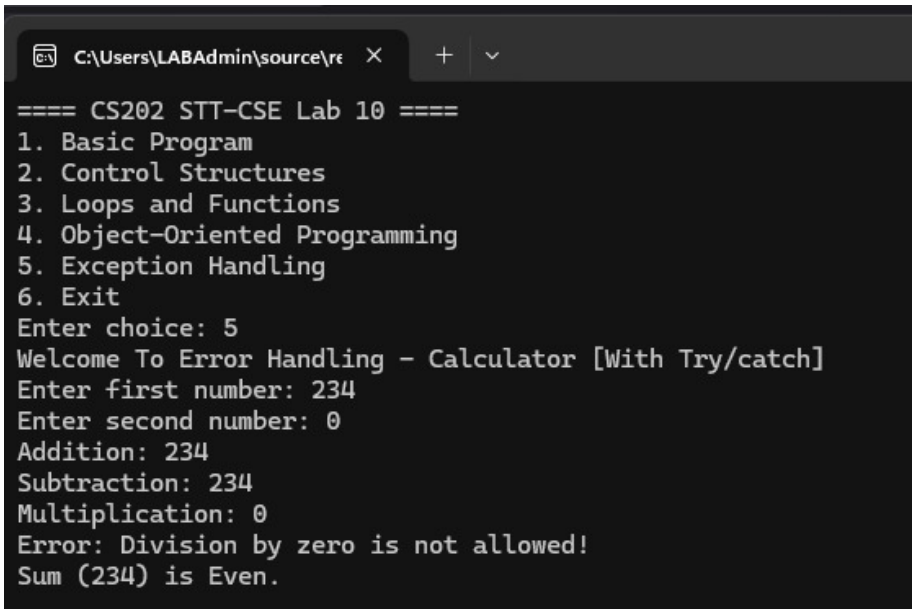


Figure 19: Exception Handling

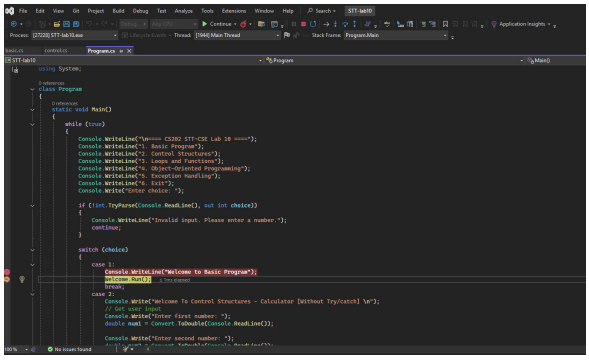
Debugging using Visual Studio Debugger: During debugging, I used Visual Studio's built-in tools to trace the flow of execution and examine the state of variables at different points in

the program. Three key operations were particularly useful: *Step-In*, *Step-Over*, and *Step-Out*.

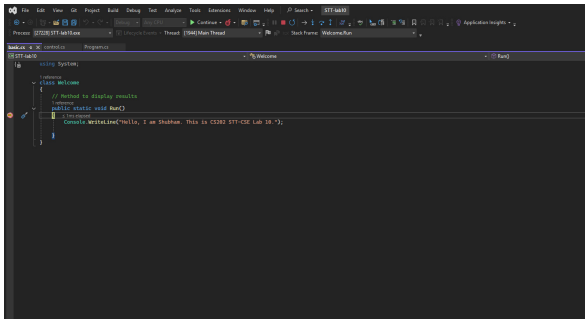
The **Step-In** command allowed me to enter into a function or method call line-by-line, which helped in understanding the internal execution of called methods. The **Step-Over** command executed the current line and moved to the next, skipping over the internals of any function calls. The **Step-Out** command completed execution of the current method and returned to the calling method.

- Basic Program:

1. Step-in:



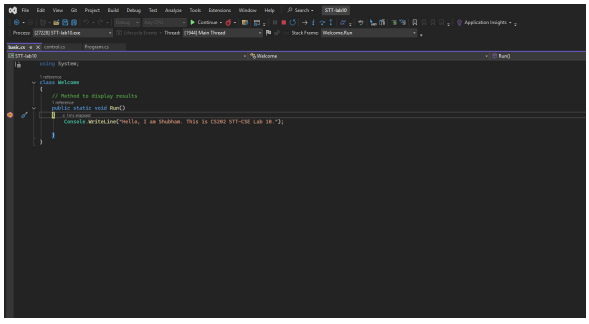
(a) Breakpoint



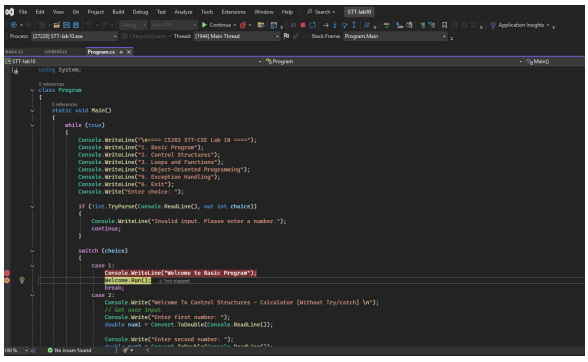
(b) Step-in

Figure 20: Step-in Result

2. Step-out:



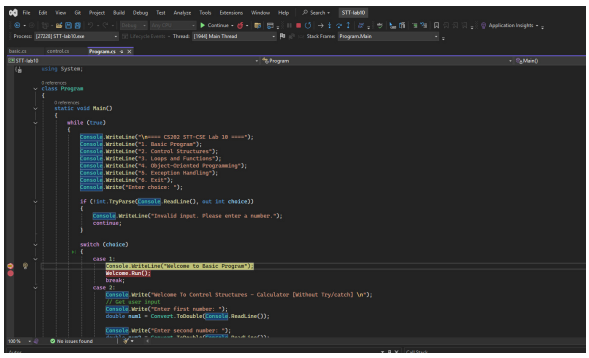
(a) Breakpoint



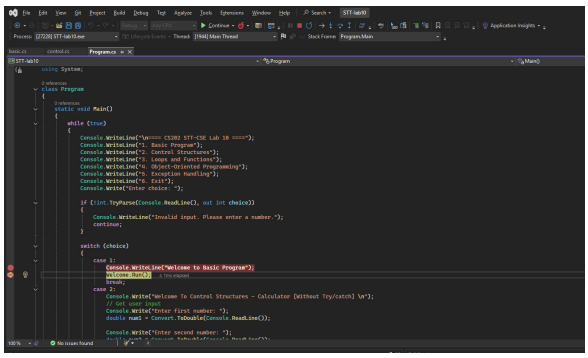
(b) Step-out

Figure 21: Step-out Result

3. Step-over:



(a) Breakpoint

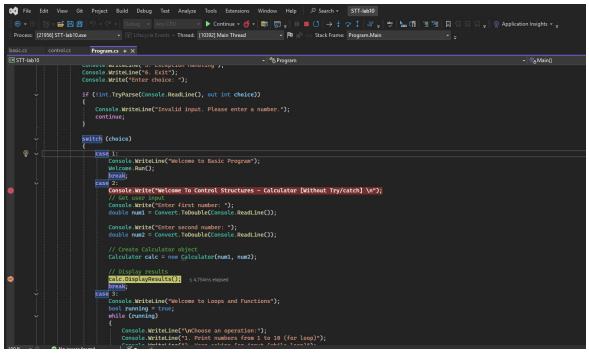


(b) Step-over

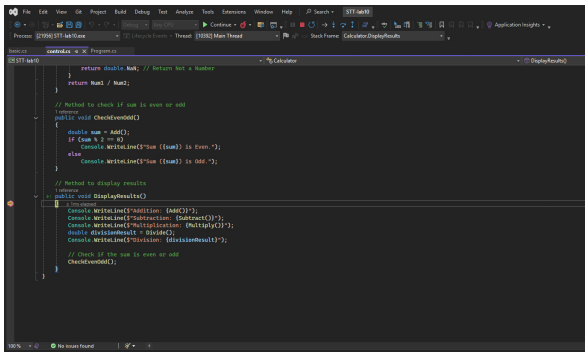
Figure 22: Step-over Result

- Control Structures:

1. Step-in:



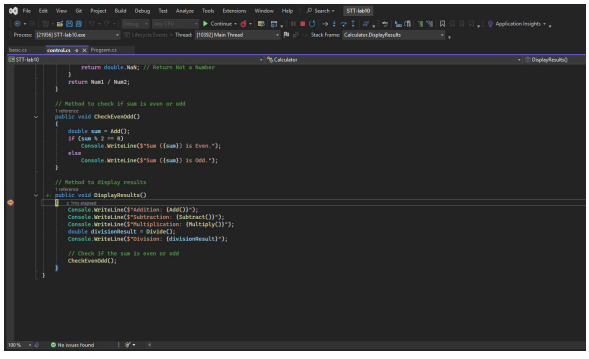
(a) Breakpoint



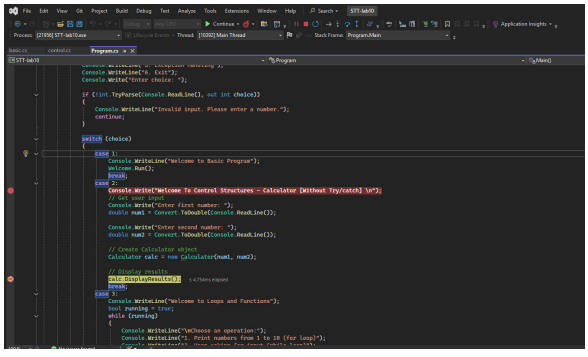
(b) Step-in

Figure 23: Step-in Result

2. Step-out:



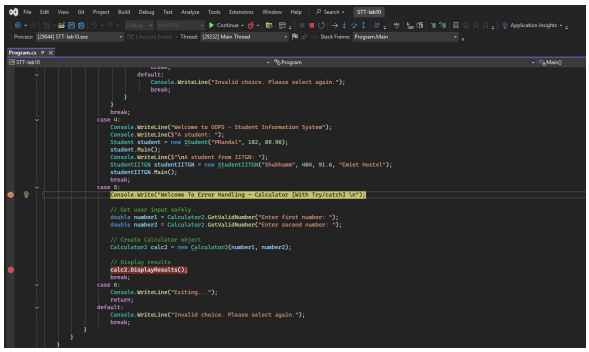
(a) Breakpoint



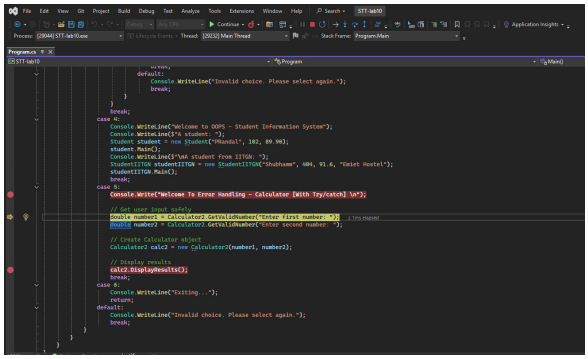
(b) Step-out

Figure 24: Step-out Result

3. Step-over:



(a) Breakpoint

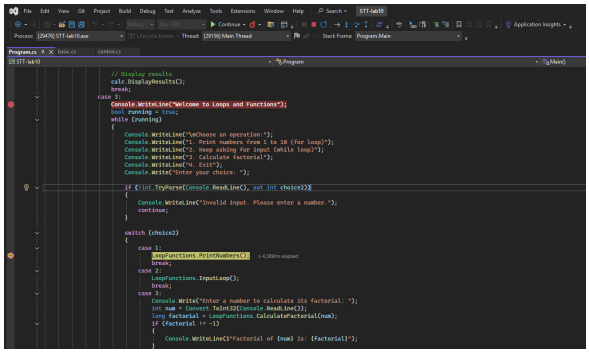


(b) Step-over

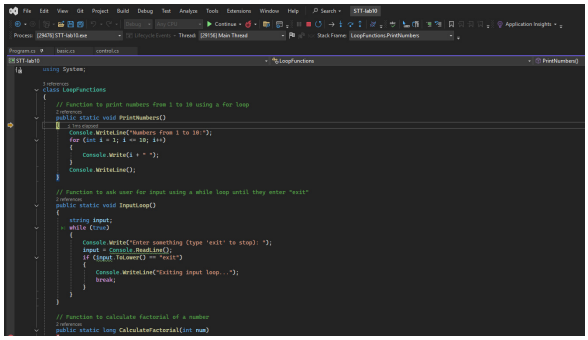
Figure 25: Step-over Result

- Loops and Functions:

1. Step-in:



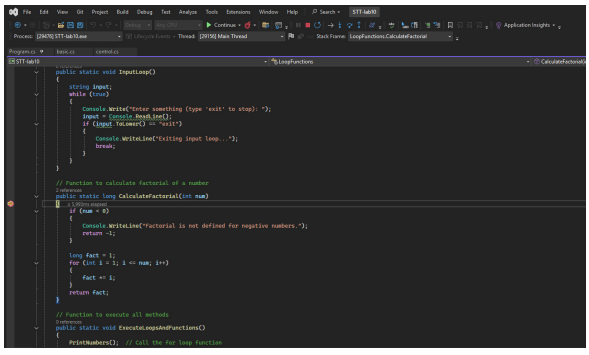
(a) Breakpoint



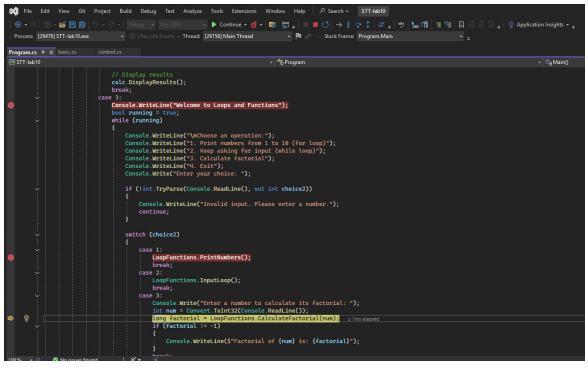
(b) Step-in

Figure 26: Step-in Result

2. Step-out:



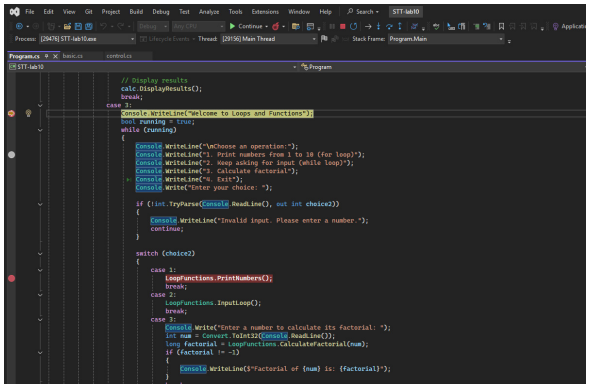
(a) Breakpoint



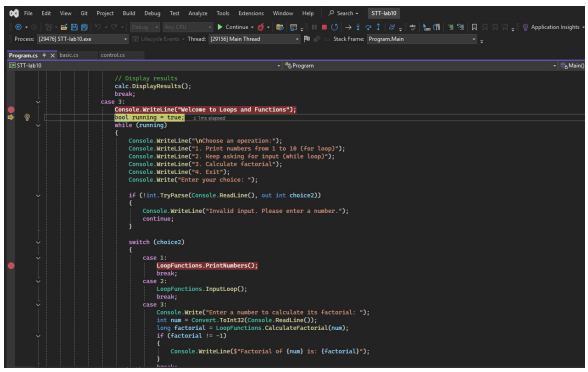
(b) Step-out

Figure 27: Step-out Result

3. Step-over:



(a) Breakpoint

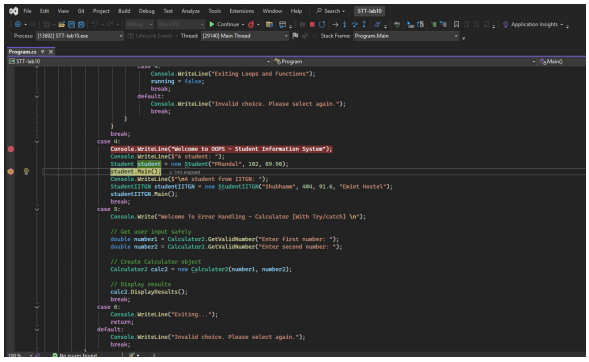


(b) Step-over

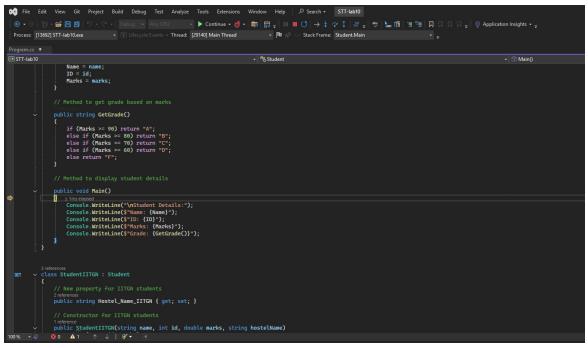
Figure 28: Step-over Result

- OOP:

1. Step-in:



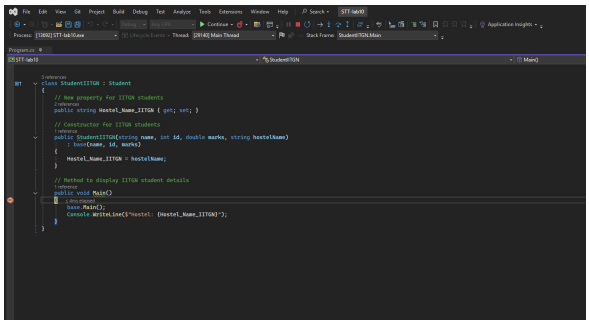
(a) Breakpoint



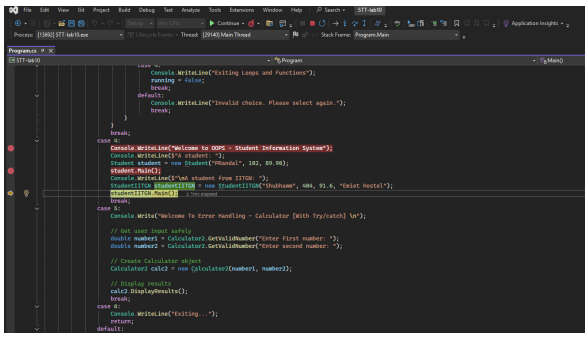
(b) Step-in

Figure 29: Step-in Result

2. Step-out:



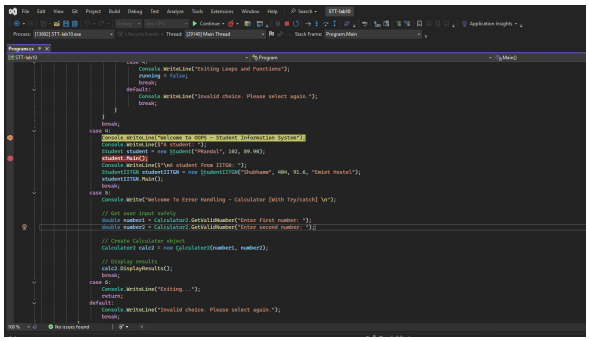
(a) Breakpoint



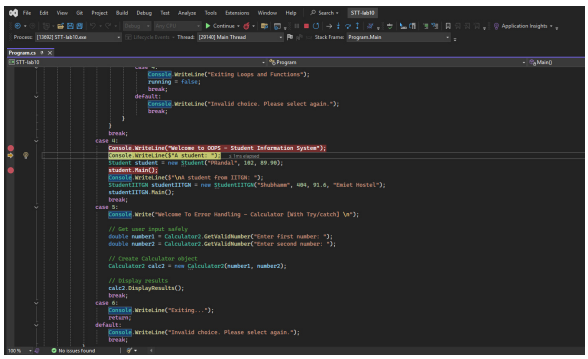
(b) Step-out

Figure 30: Step-out Result

3. Step-over:



(a) Breakpoint

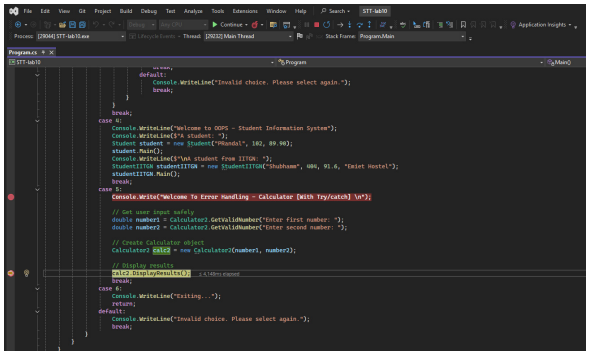


(b) Step-over

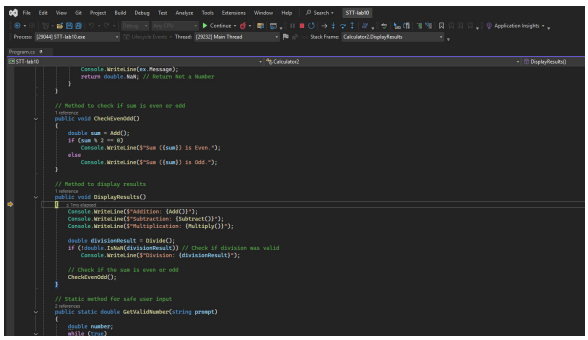
Figure 31: Step-over Result

- Exception Handling:

1. Step-in:



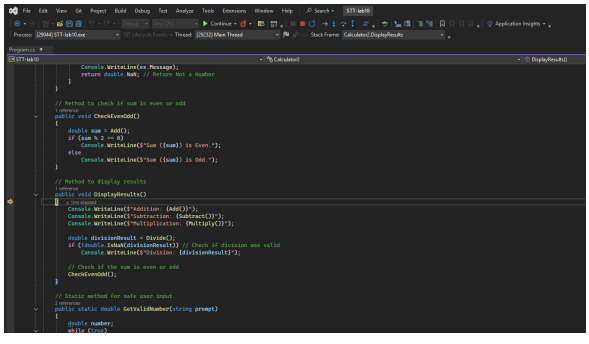
(a) Breakpoint



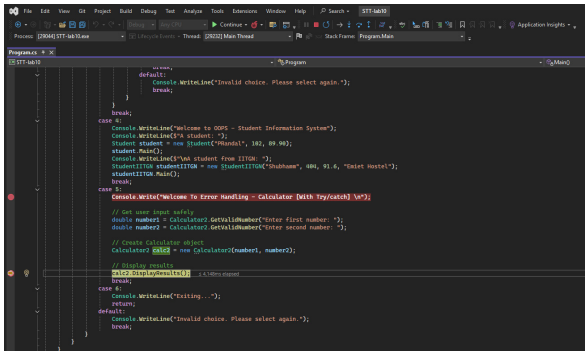
(b) Step-in

Figure 32: Step-in Result

2. Step-out:



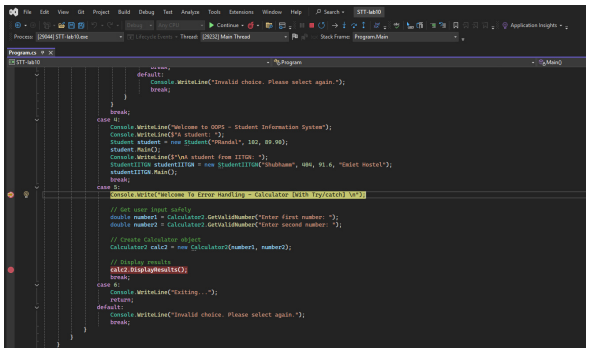
(a) Breakpoint



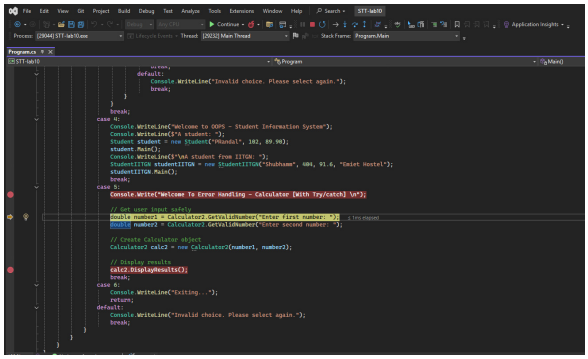
(b) Step-out

Figure 33: Step-out Result

3. Step-over:



(a) Breakpoint



(b) Step-over

Figure 34: Step-over Result

3.6.2 Observations

I observed that C# offered a familiar syntax and structure, closely aligned with languages like Java. The lab exercises were straightforward and served more as a platform to get comfortable with the Visual Studio environment and C#-specific conventions. The IDE's support for code navigation, suggestions, and debugging tools made the process efficient, especially when working across multiple source files.

3.6.3 Key Insights

One of the main takeaways was realizing how C# blends object-oriented elements with.NET-specific constructs so that modular and maintainable code is promoted. Although the actual programming logic was standard, I learned about how C# processes user input, type safety, and exception handling. Additionally, the ability to visually trace program execution using breakpoints and step functions in Visual Studio proved to be a powerful asset during testing and debugging.

3.6.4 Comparisons

In comparison to coding I had done before in Python or C, this C# programming was unique due to its formal approach and IDE-based workflow. Although Python supports rapid prototyping, I realized that more formal setup was needed in C# but with superior debugging and IDE support. In comparison to C, C# abstracted away most of the low-level issues and facilitated development at a faster rate with features such as automatic memory management and exception handling.

3.7 Discussion and Conclusion

3.7.1 Challenges Faced

One of the challenges I faced was that I struggled a bit with setting up classes and inheritance in C#, as the syntax and structure were slightly different from what I was used to in other languages.

3.7.2 Reflection

Although the actual programming work was relatively straightforward, the lab provided a good practice in using C#'s syntax, idioms, and tooling. The modular design worked well to help keep things separate and understandable. The necessity to debug with Visual Studio's step-through features also helped me get a sense of how deeply integrated the IDE is with the language runtime.

3.7.3 Lessons Learned

I learned how to create and execute a C# console application with Visual Studio, and how to code cleanly, neatly, and following object-oriented practices. I also learned how to use exceptions effectively to avoid program crashes. Most importantly, I understood the value of modular code by splitting logic across multiple files and calling them from a main driver script, which made development and testing more manageable.

3.7.4 Summary

Overall, this lab gave me hands-on experience with C# programming and the Visual Studio development environment. I completed tasks involving syntax basics, control structures, loops, functions, classes, inheritance, exception handling, and debugging. By organizing the code into separate files and using a main program to invoke each module, I was able to understand the structure of real-world applications. The lab was a comprehensive introduction to developing console applications using C# and .NET .

3.8 Files and Code Location

All files and code related to this lab can be found in the following Github repo link:

`https://github.com/shubham-agrawal04/STT-Assignment-3`