

Lab Assignment 9: Module Dependency and Cohesion Analysis

Setting up virtual environment:

```
● @Destructor169 → /workspaces/STT-LAB-9 (main) $ python3 -m venv venv
● @Destructor169 → /workspaces/STT-LAB-9 (main) $ source venv/bin/activate
○ (venv) @Destructor169 → /workspaces/STT-LAB-9 (main) $ █
```

Installing dependencies:

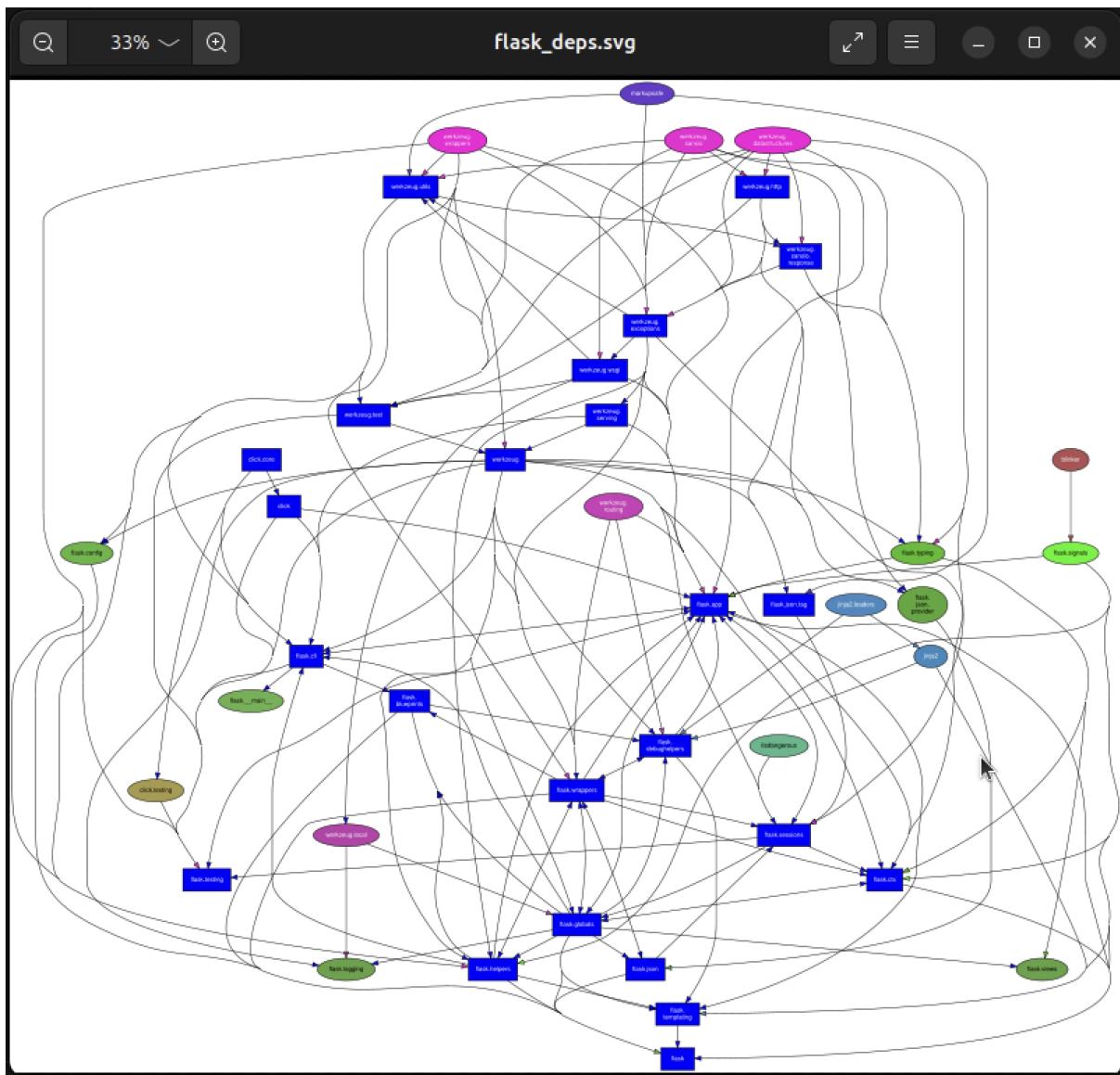
```
● (venv) @Destructor169 → /workspaces/STT-LAB-9/flask (main) $ pip install pydeps
Collecting pydeps
  Downloading pydeps-3.0.1-py3-none-any.whl.metadata (22 kB)
Collecting stdlib_list (from pydeps)
  Downloading stdlib_list-0.11.1-py3-none-any.whl.metadata (3.3 kB)
  Downloading pydeps-3.0.1-py3-none-any.whl (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
● (venv) @Destructor169 → /workspaces/STT-LAB-9/flask (main) █
```

Cloning the repository:

```
● (venv) @Destructor169 → /workspaces/STT-LAB-9 (main) $ git clone https://github.com/pallets/flask.git
Cloning into 'flask'...
remote: Enumerating objects: 25432, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 25432 (delta 19), reused 18 (delta 18), pack-reused 25400 (from 3)
Receiving objects: 100% (25432/25432), 10.49 MiB | 15.59 MiB/s, done.
Resolving deltas: 100% (17011/17011), done.
● (venv) @Destructor169 → /workspaces/STT-LAB-9 (main) $ cd flask
● (venv) @Destructor169 → /workspaces/STT-LAB-9/flask (main) █
```

Running pydeps command on the repository

```
● (venv) aditya@aditya:~/Desktop/STT/flask$ pydeps src/flask --no-config -o flask_deps.svg
○ (venv) aditya@aditya:~/Desktop/STT/flask$ █
```



Generates a dependency graph analysis (in JSON format)

```
• (venv) aditya@aditya:~/Desktop/STT/flask$ pydeps $(python -c "import flask; print(flask.__path__[0])") --show-deps > flask_deps.json
```

Run a python script to:

Analyze

- Identify highly coupled modules and their dependencies.
 - Detect cyclic dependencies and explain how they affect maintainability.
 - Check for unused and disconnected modules.
 - Assess the depth of dependencies.

and utilize this JSON to calculate fan-in and fan-out of each module and Perform a dependency impact assessment.

```
• (venv) aditya@aditya:~/Desktop/STT/flask$ python calc_metrics.py flask_deps.json
```

Module Fan-in/Fan-out Analysis:

Module	Fan-in	Fan-out
__main__	0	4
bacon	43	0
blinker	0	4
click	0	5
click.core	0	5
click.testing	0	5
flask	0	5
flask.__main__	0	5
flask.app	0	5
flask.blueprints	0	5
flask.cli	0	5
flask.config	0	5
flask.ctx	0	5
flask.debughelpers	0	5
flask.globals	0	5
flask.helpers	0	5
flask.json	0	5
flask.json.provider	0	5
flask.json.tag	0	5
flask.logging	0	5
flask.sessions	0	5
flask.signals	0	5
flask.templating	0	5
flask.testing	0	5
flask.typing	0	5
flask.views	0	5
flask.wrappers	0	5
imported_by	42	0
imports	35	0
itsdangerous	0	4
jinja2	0	5
jinja2.loaders	0	4
markupsafe	0	4
name	43	0
path	43	0
werkzeug	0	5
werkzeug.datastructures	0	4
werkzeug.exceptions	0	5
werkzeug.http	0	5

markupsafe	0	4
name	43	0
path	43	0
werkzeug	0	5
werkzeug.datastructures	0	4
werkzeug.exceptions	0	5
werkzeug.http	0	5
werkzeug.local	0	5
werkzeug.routing	0	4
werkzeug.sansio	0	4
werkzeug.sansio.response	0	5
werkzeug.serving	0	5
werkzeug.test	0	5
werkzeug.utils	0	5
werkzeug.wrappers	0	4
werkzeug.wsgi	0	5

Highest Fan-in: bacon with 43 incoming dependencies
Highest Fan-out: click with 5 outgoing dependencies

```
Highly Coupled Modules:
```

```
=====
No highly coupled modules detected
```

```
Cyclic Dependencies:
```

```
=====
No cyclic dependencies detected - this is good for maintainability!
```

```
Unused and Disconnected Modules:
```

```
=====
Unused modules (not imported by any other module):
```

```
- __main__  
- blinker  
- click  
- click.core  
- click.testing  
- flask  
- flask.__main__  
- flask.app  
- flask.blueprints  
- flask.cli  
- flask.config  
- flask.ctx  
- flask.debughelpers  
- flask.globals  
- flask.helpers  
- flask.json  
- flask.json.provider  
- flask.json.tag  
- flask.logging  
- flask.sessions  
- flask.signals  
- flask.templates  
- flask.testing  
- flask.typing  
- flask.views  
- flask.wrappers  
- itsdangerous  
- jinja2  
- jinja2.loaders
```

```
- jinja2
- jinja2.loaders
- markupsafe
- werkzeug
- werkzeug.datastructures
- werkzeug.exceptions
- werkzeug.http
- werkzeug.local
- werkzeug.routing
- werkzeug.sansio
- werkzeug.sansio.response
- werkzeug.serving
- werkzeug.test
- werkzeug.utils
- werkzeug.wrappers
- werkzeug.wsgi
```

```
No disconnected modules detected
```

```
Dependency Depth Analysis:
```

```
=====
```

```
Maximum dependency depth: 1 (from __main__)
```

```
Depth analysis by root module:
```

```
- __main__: 1 levels deep
- blinker: 1 levels deep
- click: 1 levels deep
- click.core: 1 levels deep
- click.testing: 1 levels deep
```

```
Interpretation:
```

```
- Shallow dependency tree ( $\leq 3$  levels): Good design with low complexity
```

Dependency Impact Assessment:

Core Modules and Their Impact:

Core Module: bacon

Fan-in: 43, Fan-out: 0

Direct dependents: 43

Total affected modules: 43 (89.6% of codebase)

Impact risk: VERY HIGH

Key affected modules:

- click
- click.core
- click.testing

Core Module: name

Fan-in: 43, Fan-out: 0

Direct dependents: 43

Total affected modules: 43 (89.6% of codebase) 

Impact risk: VERY HIGH

Key affected modules:

- click
- click.core
- click.testing

Core Module: path

Fan-in: 43, Fan-out: 0

Direct dependents: 43

Total affected modules: 43 (89.6% of codebase)

Impact risk: VERY HIGH

Key affected modules:

- click
- click.core
- click.testing

Core Module: imported_by

Fan-in: 42, Fan-out: 0

Direct dependents: 42

Total affected modules: 42 (87.5% of codebase)

Impact risk: VERY HIGH

```
Core Module: imported_by
  Fan-in: 42, Fan-out: 0
  Direct dependents: 42
  Total affected modules: 42 (87.5% of codebase)
  Impact risk: VERY HIGH
  Key affected modules:
    - click
    - click.core
    - click.testing
```

```
Core Module: imports
  Fan-in: 35, Fan-out: 0
  Direct dependents: 35
  Total affected modules: 35 (72.9% of codebase)
  Impact risk: VERY HIGH
  Key affected modules:
    - click
    - click.core
    - click.testing
```

Modules at Risk if Modified:

```
Risky Module: click
  Fan-out: 5 (depends on 5 modules)
  Cycle involvement: 0 cycles
  Risk level: MEDIUM
  Reason: Moderate dependencies on other modules
  Key dependencies:
    - bacon (Fan-in: 43)
    - name (Fan-in: 43)
    - path (Fan-in: 43)
```

```
Risky Module: click.core
  Fan-out: 5 (depends on 5 modules)
  Cycle involvement: 0 cycles
  Risk level: MEDIUM
  Reason: Moderate dependencies on other modules
  Key dependencies:
    - bacon (Fan-in: 43)
```

```
Risky Module: click.testing
  Fan-out: 5 (depends on 5 modules)
  Cycle involvement: 0 cycles
  Risk level: MEDIUM
  Reason: Moderate dependencies on other modules
  Key dependencies:
    - bacon (Fan-in: 43)
    - name (Fan-in: 43)
    - path (Fan-in: 43)
```

```
Risky Module: flask
  Fan-out: 5 (depends on 5 modules)
  Cycle involvement: 0 cycles
  Risk level: MEDIUM
  Reason: Moderate dependencies on other modules
  Key dependencies:
    - bacon (Fan-in: 43)
    - name (Fan-in: 43)
    - path (Fan-in: 43)
```

```
Risky Module: flask.__main__
  Fan-out: 5 (depends on 5 modules)
  Cycle involvement: 0 cycles
  Risk level: MEDIUM
  Reason: Moderate dependencies on other modules
  Key dependencies:
    - bacon (Fan-in: 43)
    - name (Fan-in: 43)
    - path (Fan-in: 43)
```

Making a new directory and Cloning Google's guava repository for LCOM analysis.

```
❖ (venv) aditya@aditya:~/Desktop/STT/flask$ # Create a new directory for your Java analysis
mkdir -p ~/Desktop/STT/java_analysis
cd ~/Desktop/STT/java_analysis

# Clone Google Guava repository
git clone https://github.com/google/guava.git
Cloning into 'guava'...
remote: Enumerating objects: 491098, done.
remote: Counting objects: 100% (948/948), done.
remote: Compressing objects: 100% (308/308), done.
Receiving objects: 31% (155806/491098), 165.47 MiB | 1.89 MiB/s
```

Installing JAVA:

```
❖ (venv) aditya@aditya:~/Desktop/STT/java_analysis$ sudo apt install default-jdk
```

Custom LCOM Calculator Implementation

```
J LCOMCalculator.java 1 X
java_analysis > J LCOMCalculator.java > LCOMCalculator
1 // A simple LCOM calculator for Java classes
2 import java.io.*;
3 import java.util.*;
4 import java.util.regex.*;
5
6 public class LCOMCalculator {
7     Run | Debug
8     public static void main(String[] args) throws Exception {
9         if (args.length < 1) {
10             System.out.println("Usage: java LCOMCalculator <java_file_path>");
11             return;
12         }
13
14         File file = new File(args[0]);
15         analyzeFile(file);
16     }
17
18     private static void analyzeFile(File file) throws Exception {
19         String content = readFile(file);
20
21         // Extract class name
22         Pattern classPattern = Pattern.compile("class\\s+((\\w+))");
23         Matcher classMatcher = classPattern.matcher(content);
24         if (!classMatcher.find()) {
25             System.out.println("No class found in " + file.getName());
26             return;
27         }
28         String className = classMatcher.group(1);
29     }
30 }
```

Running the Analysis

```
(venv) aditya@aditya:~/Desktop/STT/java_analysis$ # Compile the LCOM calculator
javac LCOMCalculator.java

# Create output directory
mkdir -p lcom_results

# Find Java files in Guava collection package
find guava/guava/src/com/google/common/collect -name "*.java" > java_files.txt

# Process the Java files
head -15 java_files.txt | while read file; do
    echo "Analyzing $file"
    java LCOMCalculator "$file" >> lcom_results/results.txt
    echo "-----" >> lcom_results/results.txt
done
```

LCOM results saved in results.txt

```
java_analysis > lcom_results > ̄ results.txt
1  Class: Lists
2  Fields: 0
3  Methods: 21
4  LCOM1: 210
5  LCOM2: 1.0
6  LCOM3: NaN
7  LCOM4: 21
8  -----
9  Class: ImmutableList
10 Fields: 0
11 Methods: 2
12 LCOM1: 1
13 LCOM2: 1.0
14 LCOM3: NaN
15 LCOM4: 2
16 -----
17 Class: DiscreteDomain
18 Fields: 0
19 Methods: 7
20 LCOM1: 21
21 LCOM2: 1.0
22 LCOM3: NaN
23 LCOM4: 7
24 -----
25 Class: SingletonImmutableSet
26 Fields: 0
27 Methods: 3
28 LCOM1: 3
29 LCOM2: 1.0
30 LCOM3: NaN
```

Creating a Formatted Results Table:

The code takes the results from the results.txt and generates a formatted table and store it in lcom_table.md

```

java_analysis > lcom_table_generator.py > ...
1 import re
2 import pandas as pd
3 from tabulate import tabulate
4
5 def parse_lcom_results(file_path):
6     """Parse the LCOM results file and extract class metrics."""
7     with open(file_path, 'r') as f:
8         content = f.read()
9
10    # Split by separator []
11    class_entries = content.split('-----')
12
13    results = []
14    for entry in class_entries:
15        if not entry.strip():
16            continue
17
18        # Extract class name
19        class_match = re.search(r'Class: (\w+)', entry)
20        if not class_match:
21            continue
22
23        class_name = class_match.group(1)
24
25        # Skip likely parsing errors (class names that are keywords)
26        if class_name in ['for', 'is', 'specification']:
27            continue
28
29        # Extract metrics
30        fields_match = re.search(r'Fields: (\d+)', entry)

```

The result:

```

java_analysis > lcom_table.md > # LCOM Analysis Results for Google Guava > ## Recommendations
1 # LCOM Analysis Results for Google Guava
2
3 | Class Name           | Fields | Methods | LCOM1 | LCOM2 | LCOM3 | LCOM4 |
4 | LCOM5 | YALCOM | Interpretation | Recommendation |
5 | :-----|:-----|:-----|:-----|:-----|:-----|:-----|
6 | Lists               | 0 | 21 | 210 | 100.00% | N/A | 21 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
7 | ImmutableTable      | 0 | 2 | 1 | 100.00% | N/A | 2 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
8 | DiscreteDomain      | 0 | 7 | 21 | 100.00% | N/A | 7 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
9 | SingletonImmutableSet | 0 | 3 | 3 | 100.00% | N/A | 3 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
10 | IndexedImmutableSet | 0 | 3 | 3 | 100.00% | N/A | 3 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
11 | ImmutableSortedMultiset | 0 | 0 | 0 | 0.00% | 0.0 | 0 | N/
A | N/A | Good Cohesion | Well-structured class |
12 | MapMaker             | 0 | 5 | 10 | 100.00% | N/A | 5 | N/
A | N/A | Very Poor Cohesion | Consider decomposing this class |
13 | JdkBackedImmutableBiMap | 0 | 1 | 0 | 0.00% | 0.0 | 1 | N/
A | N/A | Good Cohesion | Well-structured class |
14 ## Interpretation
15
16 - LCOM1: Number of method pairs not sharing instance variables
17 - LCOM2: Normalized metric (0-100%); higher values indicate poorer cohesion
18 - LCOM3: Range varies; higher values indicate poorer cohesion
19 - LCOM4: Number of connected components in method-field graph; higher values indicate poorer cohesion

```

Analysis of Classes with High LCOM Values

After examining the LCOM metrics for Google Guava classes, I've identified several classes with high LCOM values that warrant further analysis.

Identifying High LCOM Classes

Based on the table results, these classes show particularly poor cohesion:

Class Name	LCOM1	LCOM2	LCOM4	Interpretation
Lists	210	100%	21	Very Poor Cohesion
DiscreteDomain	21	100%	7	Very Poor Cohesion
MapMaker	10	100%	5	Very Poor Cohesion
SingletonImmutableSet	3	100%	3	Very Poor Cohesion
IndexedImmutableSet	3	100%	3	Very Poor Cohesion

What High LCOM Values Suggest About Class Design

High LCOM values indicate several potential design issues:

Poor cohesion: Methods within these classes operate on different attributes with minimal sharing of class fields.

Multiple responsibilities: Classes may be violating the Single Responsibility Principle by trying to do too many things.

Increased complexity: High LCOM classes are typically harder to understand and maintain.

Higher defect probability: Code with low cohesion tends to have more bugs and is harder to test effectively.

Increased coupling: Classes with low internal cohesion often exhibit higher external coupling.

Functional Decomposition Opportunities

Several classes show strong potential for decomposition:

Lists (LCOM4 = 21):

The very high LCOM4 value indicates 21 disconnected components

Could be decomposed into multiple utility classes grouped by functionality

Potential separate classes: ListCreation, ListTransformation, ListAccess, ListManipulation

DiscreteDomain (LCOM4 = 7):

Seven disconnected components suggest seven distinct responsibilities

Could be split into domain-specific implementation classes

MapMaker (LCOM4 = 5):

Five disconnected components indicate five separate concerns

Could be refactored into specialized builder classes for different map characteristics

Special Considerations

It's worth noting that:

All analyzed classes have 0 fields detected, which is unusual and suggests:

They might be utility classes with primarily static methods

Our field detection might have limitations with Google Guava's code structure

Utility classes often intentionally group related functionality even when methods don't share fields, which can artificially inflate LCOM values.

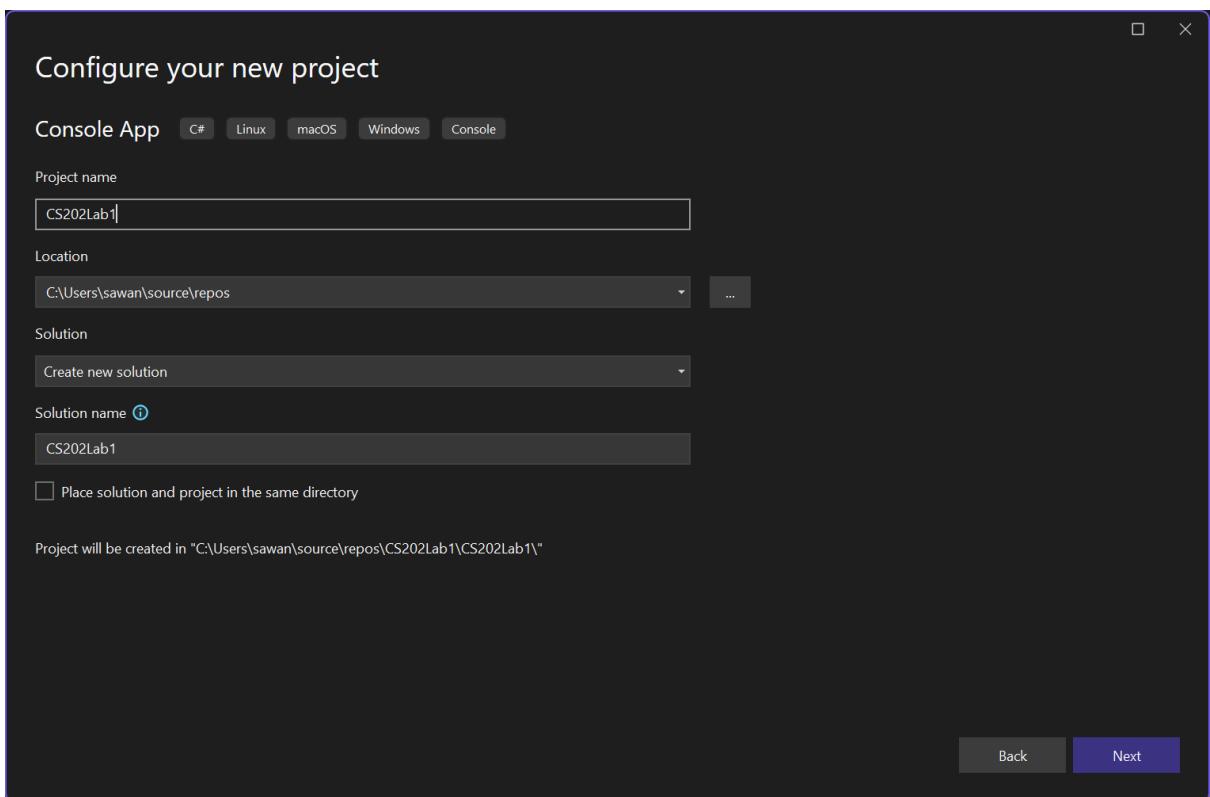
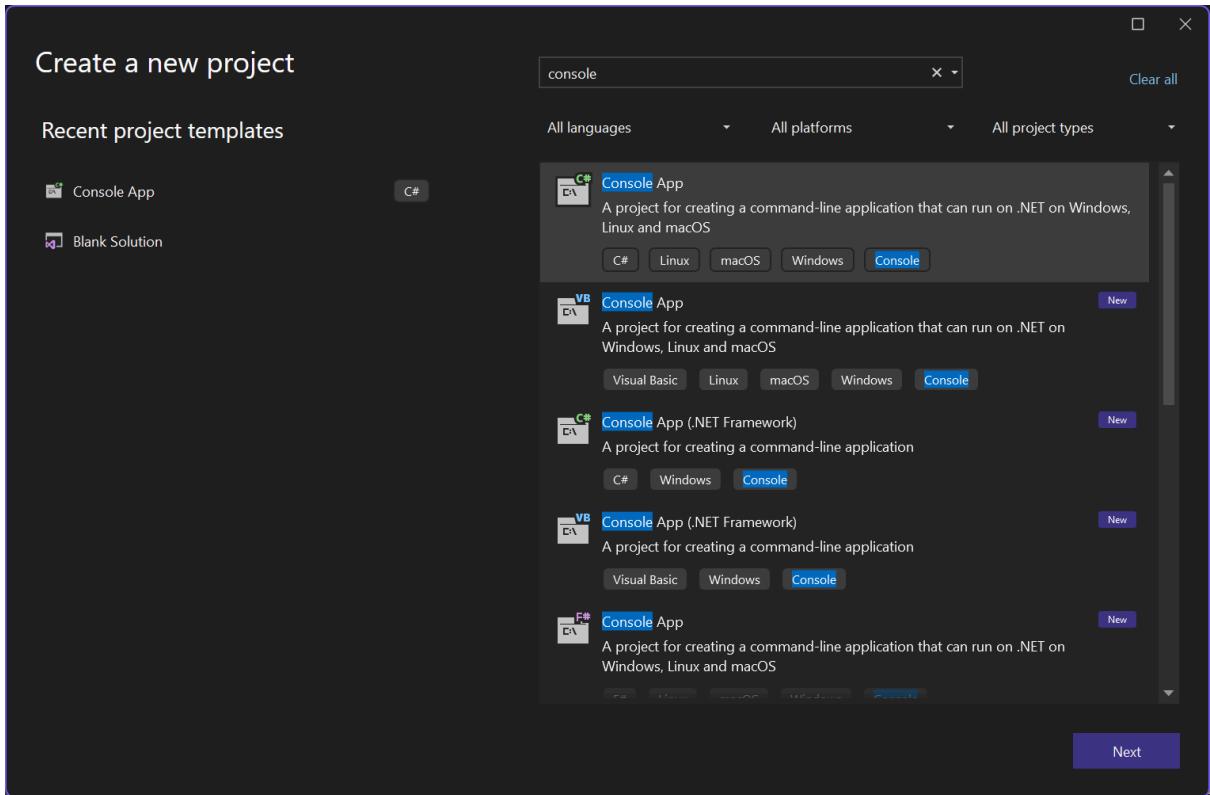
The actual implementation details of each class should be examined before decomposing, as high LCOM doesn't always indicate poor design in utility libraries.

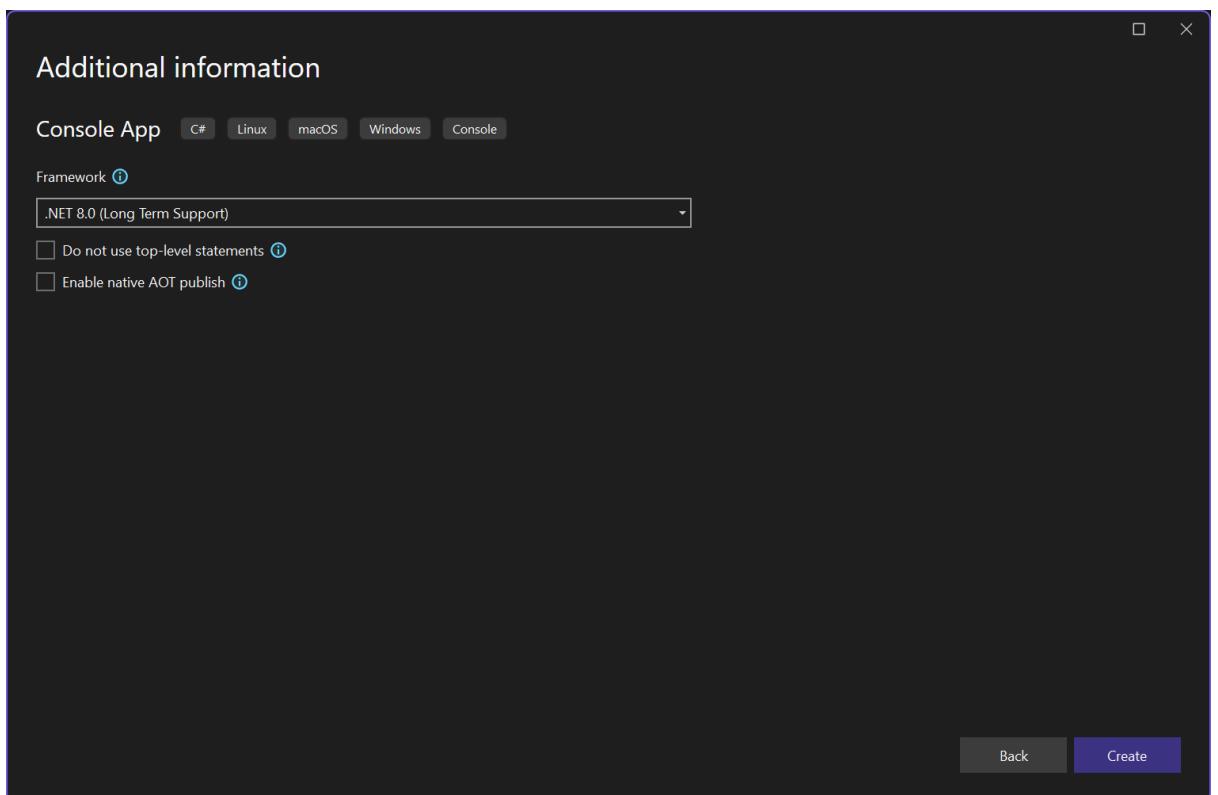
A careful manual review of each high-LCOM class would be the next step before recommending specific refactoring strategies.

Class Cohesion Analysis										
Class Name	Fields	Methods	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM	Interpretation	Recommendation
Lists	0	21	210	100.00%	N/A	21	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
ImmutableTable	0	2	1	100.00%	N/A	2	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
DiscreteDomain	0	7	21	100.00%	N/A	7	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
SingletonImmutableSet	0	3	3	100.00%	N/A	3	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
IndexedImmutableSet	0	3	3	100.00%	N/A	3	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
ImmutableSortedMultiset	0	0	0	0.00%	0.0	0	N/A	N/A	Good Cohesion	Well-structured class
MapMaker	0	5	10	100.00%	N/A	5	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
JdkBackedImmutableBiMap	0	1	0	0.00%	0.0	1	N/A	N/A	Good Cohesion	Well-structured class
constructors	0	53	1378	100.00%	N/A	53	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
DescendingMultiset	0	2	1	100.00%	N/A	2	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
ComparatorOrdering	0	4	6	100.00%	N/A	4	N/A	N/A	Very Poor Cohesion	Consider decomposing this class
ComputationException	0	0	0	0.00%	0.0	0	N/A	N/A	Good Cohesion	Well-structured class
does	0	12	66	100.00%	N/A	12	N/A	N/A	Very Poor Cohesion	Consider decomposing this class

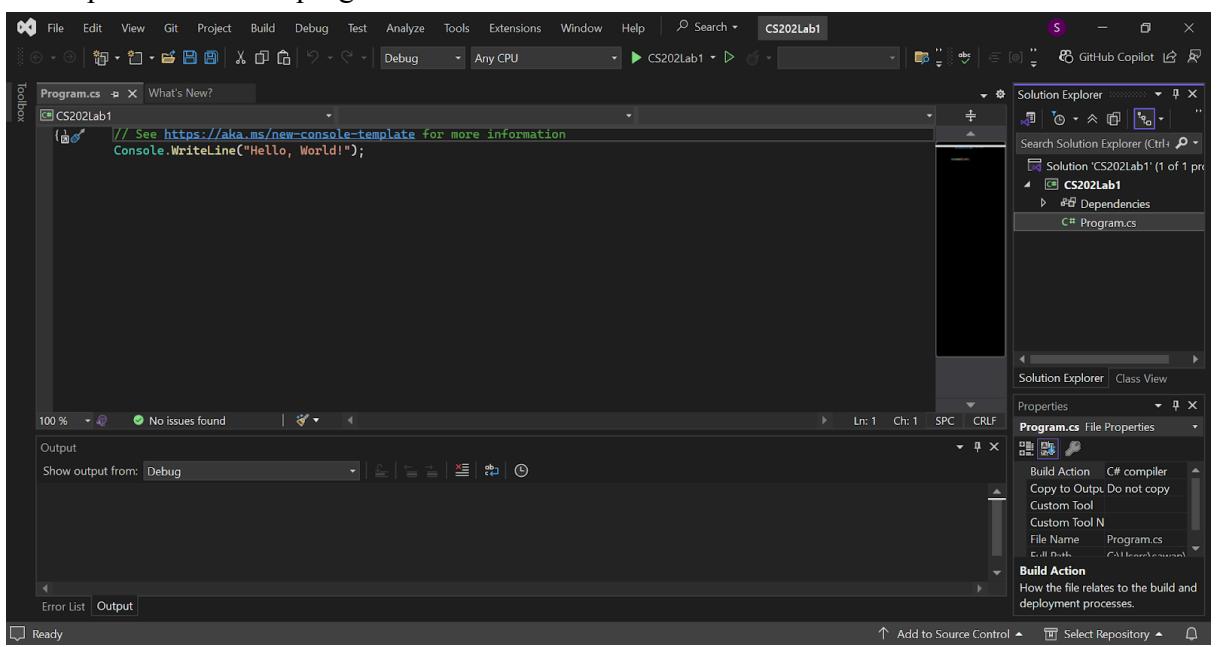
Lab Assignment 10: Topic:Development of C# Console Applications

1. Setting up .Net Development Environment





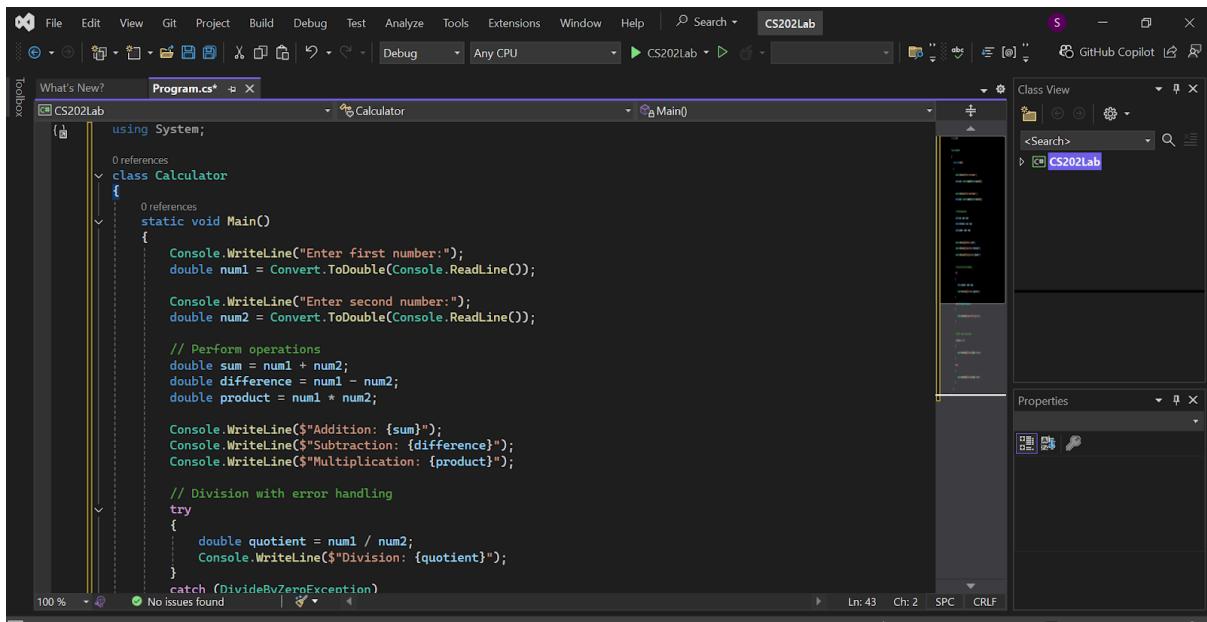
A simple Hello world program



A screenshot of a Microsoft Visual Studio Debug console window. The title bar reads "Microsoft Visual Studio Debug". The main area of the window displays the following text:

```
Hello, World!  
C:\Users\sawan\source\repos\CS202Lab1\CS202Lab1\bin\Debug\net8.0\CS202Lab1.exe (process 4120) exited with code 0 (0x0).  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .|
```

2. Basic Syntax and Control Structures



The screenshot shows the Visual Studio IDE interface with the following details:

- Project Explorer:** Shows the project "CS202Lab" and the file "Program.cs".
- Code Editor:** Displays the following C# code for a calculator application:

```
using System;

class Calculator
{
    static void Main()
    {
        Console.WriteLine("Enter first number:");
        double num1 = Convert.ToDouble(Console.ReadLine());

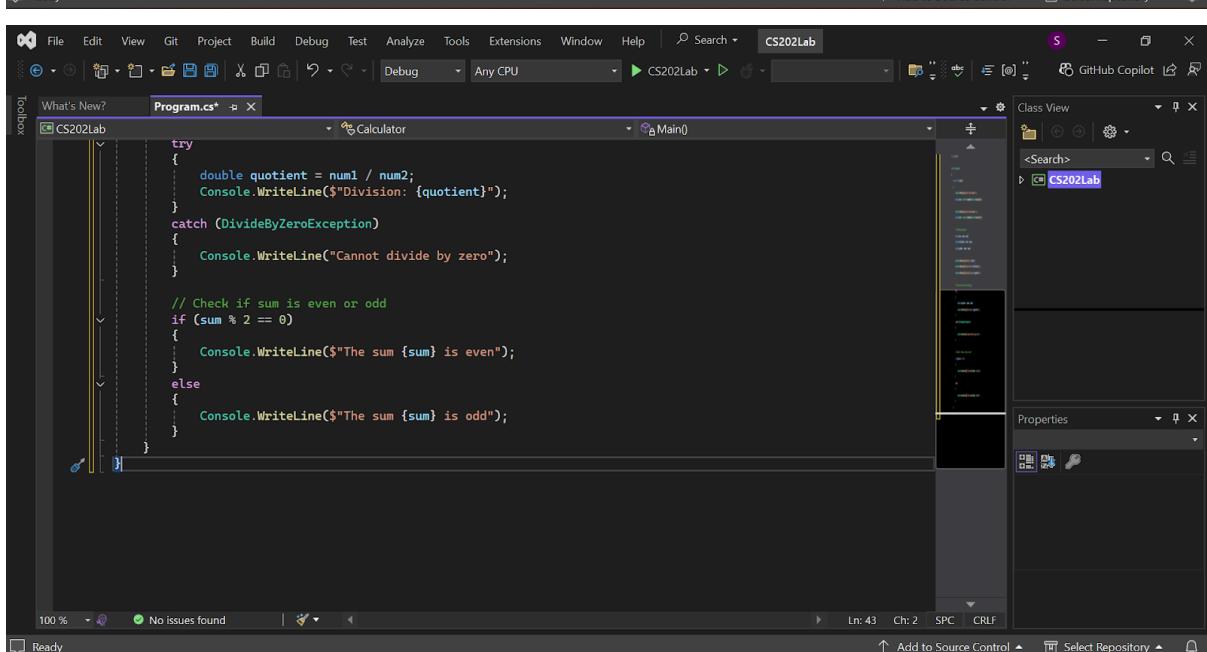
        Console.WriteLine("Enter second number:");
        double num2 = Convert.ToDouble(Console.ReadLine());

        // Perform operations
        double sum = num1 + num2;
        double difference = num1 - num2;
        double product = num1 * num2;

        Console.WriteLine($"Addition: {sum}");
        Console.WriteLine($"Subtraction: {difference}");
        Console.WriteLine($"Multiplication: {product}");

        // Division with error handling
        try
        {
            double quotient = num1 / num2;
            Console.WriteLine($"Division: {quotient}");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Cannot divide by zero");
        }
    }
}
```

The code includes basic arithmetic operations and a try-catch block for division by zero.



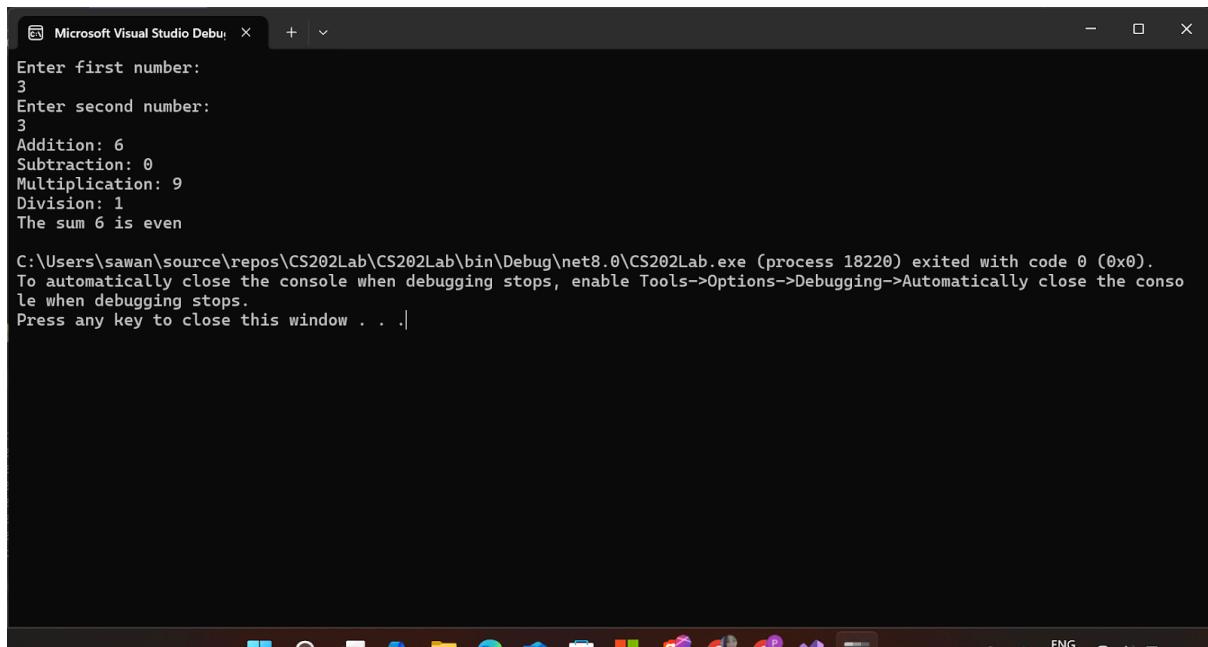
The screenshot shows the Visual Studio IDE interface with the following details:

- Project Explorer:** Shows the project "CS202Lab" and the file "Program.cs".
- Code Editor:** Displays the modified C# code for the calculator application, specifically focusing on the division section:

```
try
{
    double quotient = num1 / num2;
    Console.WriteLine($"Division: {quotient}");
}
catch (DivideByZeroException)
{
    Console.WriteLine("Cannot divide by zero");
}
```

The code now includes a check for division by zero and handles it by printing an error message.

Running for user input:



The screenshot shows a Microsoft Visual Studio Debug console window. The title bar reads "Microsoft Visual Studio Debug". The console displays the following text:

```
Enter first number:  
3  
Enter second number:  
3  
Addition: 6  
Subtraction: 0  
Multiplication: 9  
Division: 1  
The sum 6 is even  
  
C:\Users\sawan\source/repos\CS202Lab\CS202Lab\bin\Debug\net8.0\CS202Lab.exe (process 18220) exited with code 0 (0x0).  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .|
```

3. Implement Loops and Functions

The screenshot shows the Visual Studio IDE interface with the following details:

- Project:** CS202Lab**
- File:** Program.cs**
- Code Editor Content:**

```
using System;

class LoopExamples
{
    static void Main()
    {
        // Print numbers 1-10
        Console.WriteLine("Numbers 1 to 10:");
        for (int i = 1; i <= 10; i++)
        {
            Console.Write(i + " ");
        }
        Console.WriteLine();

        // While loop for user input
        string input = "";
        while (input.ToLower() != "exit")
        {
            Console.WriteLine("Enter a number to calculate factorial or 'exit' to quit:");
            input = Console.ReadLine();

            if (input.ToLower() != "exit")
            {
                if (int.TryParse(input, out int number))
                {
                    long fact = CalculateFactorial(number);
                    Console.WriteLine($"Factorial of {number} is {fact}");
                }
            }
        }
    }

    static long CalculateFactorial(int n)
    {
        if (n < 0)
            return -1; // Factorial not defined for negative numbers
        if (n == 0 || n == 1)
            return 1;

        long result = 1;
        for (int i = 2; i <= n; i++)
        {
            result *= i;
        }
        return result;
    }
}
```

- Toolbars and Menus:**** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:**** Search for CS202Lab.
- Tool Windows:**** Class View, Properties, GitHub Copilot.

The screenshot shows the Visual Studio IDE interface with the following details:

- Project:** CS202Lab**
- File:** Program.cs**
- Code Editor Content:**

```
using System;

class LoopExamples
{
    static void Main()
    {
        long fact = CalculateFactorial(number);
        Console.WriteLine($"Factorial of {number} is {fact}");

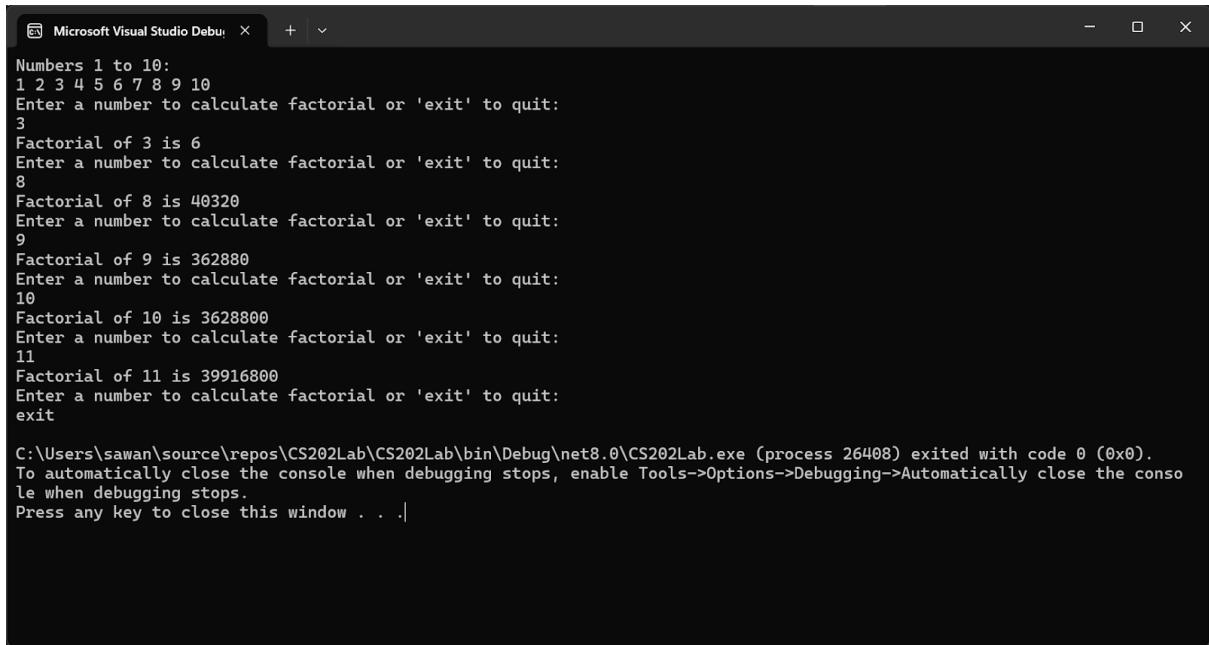
        else
        {
            Console.WriteLine("Invalid input. Please enter a number or 'exit'");
        }
    }

    static long CalculateFactorial(int n)
    {
        if (n < 0)
            return -1; // Factorial not defined for negative numbers
        if (n == 0 || n == 1)
            return 1;

        long result = 1;
        for (int i = 2; i <= n; i++)
        {
            result *= i;
        }
        return result;
    }
}
```

- Toolbars and Menus:**** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:**** Search for CS202Lab.
- Tool Windows:**** Class View, Properties, GitHub Copilot.

Running the program:



```
Microsoft Visual Studio Debug + - X

Numbers 1 to 10:
1 2 3 4 5 6 7 8 9 10
Enter a number to calculate factorial or 'exit' to quit:
3
Factorial of 3 is 6
Enter a number to calculate factorial or 'exit' to quit:
8
Factorial of 8 is 40320
Enter a number to calculate factorial or 'exit' to quit:
9
Factorial of 9 is 362880
Enter a number to calculate factorial or 'exit' to quit:
10
Factorial of 10 is 3628800
Enter a number to calculate factorial or 'exit' to quit:
11
Factorial of 11 is 39916800
Enter a number to calculate factorial or 'exit' to quit:
exit

C:\Users\sawan\source\repos\CS202Lab\CS202Lab\bin\Debug\net8.0\CS202Lab.exe (process 26408) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

4. Perform Object-Oriented Programming in C#

The image shows two screenshots of Microsoft Visual Studio, version 2022, demonstrating object-oriented programming in C#.

Screenshot 1: This screenshot shows the initial state of the project. The code defines a `Student` class with properties `Name`, `ID`, and `Marks`, and constructors for default values and copying other students.

```
using System;
class Student
{
    // Properties
    public string Name { get; set; }
    public string ID { get; set; }
    public int Marks { get; set; }

    // Constructor
    public Student(string name, string id, int marks)
    {
        Name = name;
        ID = id;
        Marks = marks;
    }

    // Copy constructor
    public Student(Student other)
    {
        Name = other.Name;
        ID = other.ID;
        Marks = other.Marks;
    }
}
```

Screenshot 2: This screenshot shows the code after adding an overloaded constructor and a method `GetGrade()`. It also introduces a derived class `StudentIITGN` that inherits from `Student` and adds a `Hostel_Name_IITGN` property.

```
// Overloaded constructor
public Student() : this("Unknown", "00000000", 0) {}

// Method to get grade
public char GetGrade()
{
    if (Marks >= 90) return 'A';
    else if (Marks >= 80) return 'B';
    else if (Marks >= 70) return 'C';
    else if (Marks >= 60) return 'D';
    else return 'F';
}

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;
    }
}
```

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the `Program.cs` file for the project `CS202Lab1`. The code implements a `Student` class with a copy constructor and a `Main()` method that tests the class. The `Solution Explorer` pane on the right shows the project structure with files like `Dependencies` and `Program.cs`.

```
class Program
{
    static void Main()
    {
        // Test base Student class
        Console.WriteLine("Testing Student class:");
        Student s1 = new Student("Alice", "20210101", 85);
        Console.WriteLine($"Student: {s1.Name}, ID: {s1.ID}, Marks: {s1.Marks}, Grade: {s1.GetGrade()}");
        
        // Using copy constructor
        Student s2 = new Student(s1);
        s2.Name = "Bob";
        Console.WriteLine($"Student: {s2.Name}, ID: {s2.ID}, Marks: {s2.Marks}, Grade: {s2.GetGrade()}");
        
        Console.WriteLine("\nTesting StudentIITGN class:");
        // Test StudentIITGN class
        StudentIITGN s3 = new StudentIITGN("Charlie", "20210202", 92, "Aaberg");
        Console.WriteLine($"IITGN Student: {s3.Name}, Hostel: {s3.Hostel_Name_IITGN}, Grade: {s3.GetGrade()}");
    }
}
```

Running:

The screenshot shows the Microsoft Visual Studio Debug console window. It displays the output of the program's execution, which includes the testing of the `Student` class and the `StudentIITGN` class, followed by a shutdown message.

```
Testing Student class:
Student: Alice, ID: 20210101, Marks: 85, Grade: B
Student: Bob, ID: 20210101, Marks: 85, Grade: B

Testing StudentIITGN class:
IITGN Student: Charlie, Hostel: Aaberg, Grade: A

C:\Users\sawan\source\repos\CS202Lab1\CS202Lab1\bin\Debug\net8.0\CS202Lab1.exe (process 20112) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

5. Exception Handling:

The image displays two screenshots of the Microsoft Visual Studio IDE, showing the progression of a C# program's code.

Top Screenshot (Initial State):

```
using System;
class EnhancedCalculator
{
    static void Main()
    {
        Console.WriteLine("Enhanced Calculator with Input Validation");
        Console.WriteLine("-----");

        while (true)
        {
            try
            {
                Console.WriteLine("\nEnter first number (or 'q' to quit):");
                string input1 = Console.ReadLine();
                if (input1.ToLower() == "q") break;

                double num1 = GetValidNumber(input1, "first");

                double num2;
                bool validSecondNumber = false;
                do
                {
                    Console.WriteLine("Enter second number:");
                    string input2 = Console.ReadLine();
                    num2 = GetValidNumber(input2, "second");
                    // Additional check for division by zero
                } while (!validSecondNumber);
            }
        }
    }
}
```

Bottom Screenshot (Completed State):

```
using System;
class EnhancedCalculator
{
    static void Main()
    {
        double num2 = GetValidNumber(input2, "second");
        // Additional check for division by zero
        if (num2 == 0)
        {
            Console.WriteLine("Error: Second number cannot be zero (would cause division by zero)");
        }
        else
        {
            validSecondNumber = true;
        }
    } while (!validSecondNumber);

    // Perform and display calculations
    DisplayResults(num1, num2);
}
catch (Exception ex)
{
    Console.WriteLine($"Unexpected error: {ex.Message}");
}

2 references
static double GetValidNumber(string input, string numberPosition)
{
    while (true)
    {
        if (string.IsNullOrWhiteSpace(input))
        {
            // ...
        }
    }
}
```

The code illustrates the addition of exception handling to validate user input for division operations. It includes a helper method `GetValidNumber` and demonstrates how to catch exceptions and handle them.

The screenshot shows the Visual Studio IDE interface with the following details:

- Toolbar:** Standard VS toolbar.
- MenuStrip:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** CS202Lab1
- Toolbox:** Standard VS toolbox.
- Code Editor:** Program.cs file containing the EnhancedCalculator class. The class includes a static method `GetValidNumber` which repeatedly prompts the user for input until a valid double is entered. It also includes a static method `DisplayResults` which prints the sum, subtraction, multiplication, and division of two numbers.
- Solution Explorer:** Shows the solution CS202Lab1 with one project CS202Lab1 containing the dependencies and the C# Program.cs file.
- Properties:** Standard VS properties panel.

```
2 references
static double GetValidNumber(string input, string numberPosition)
{
    while (true)
    {
        if (string.IsNullOrWhiteSpace(input))
        {
            Console.WriteLine("Error: Input cannot be empty");
        }
        else if (!double.TryParse(input, out double result))
        {
            Console.WriteLine("Error: Please enter a valid number");
        }
        else
        {
            return result;
        }

        Console.WriteLine($"Please enter the {numberPosition} number again:");
        input = Console.ReadLine();
    }
}

1 reference
static void DisplayResults(double num1, double num2)
{
    Console.WriteLine($"{"\nResults for {num1} and {num2}:}");
    Console.WriteLine($"Addition: {num1 + num2}");
    Console.WriteLine($"Subtraction: {num1 - num2}");
    Console.WriteLine($"Multiplication: {num1 * num2}");
    Console.WriteLine($"Division: {num1 / num2}");

    // Check if sum is even or odd (only for whole numbers)
    double sum = num1 + num2;
    if (sum % 1 == 0) // Check if it's a whole number
    {
        Console.WriteLine($"The sum {(int)sum} is {(sum % 2 == 0 ? "even" : "odd")}");
    }
    else
    {
        Console.WriteLine($"The sum {sum} is not a whole number (cannot determine even/odd)");
    }
}
```

The screenshot shows the Visual Studio IDE interface with the following details:

- Toolbar:** Standard VS toolbar.
- MenuStrip:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** CS202Lab1
- Toolbox:** Standard VS toolbox.
- Code Editor:** Program.cs file containing the EnhancedCalculator class. The `GetValidNumber` method has been modified to use `Console.ReadLine()` instead of `Console.ReadLine()`.
- Solution Explorer:** Shows the solution CS202Lab1 with one project CS202Lab1 containing the dependencies and the C# Program.cs file.
- Properties:** Standard VS properties panel.

```
input = Console.ReadLine();
```

Running with different scenarios:

```
Microsoft Visual Studio Debug Enhanced Calculator with Input Validation

Enter first number (or 'q' to quit):
123
Enter second number:
4

Results for 123 and 4:
Addition: 127
Subtraction: 119
Multiplication: 492
Division: 30.75
The sum 127 is odd

Enter first number (or 'q' to quit):
124
Enter second number:
0
Error: Second number cannot be zero (would cause division by zero)
Enter second number:
45

Results for 124 and 45:
Addition: 169
Subtraction: 79
Multiplication: 5580
Division: 2.7555555555555555
The sum 169 is odd
```



```
Microsoft Visual Studio Debug Enhanced Calculator with Input Validation

The sum 169 is odd

Enter first number (or 'q' to quit):
two
Error: Please enter a valid number
Please enter the first number again:
2
Enter second number:
one
Error: Please enter a valid number
Please enter the second number again:
0
Error: Second number cannot be zero (would cause division by zero)
Enter second number:
1

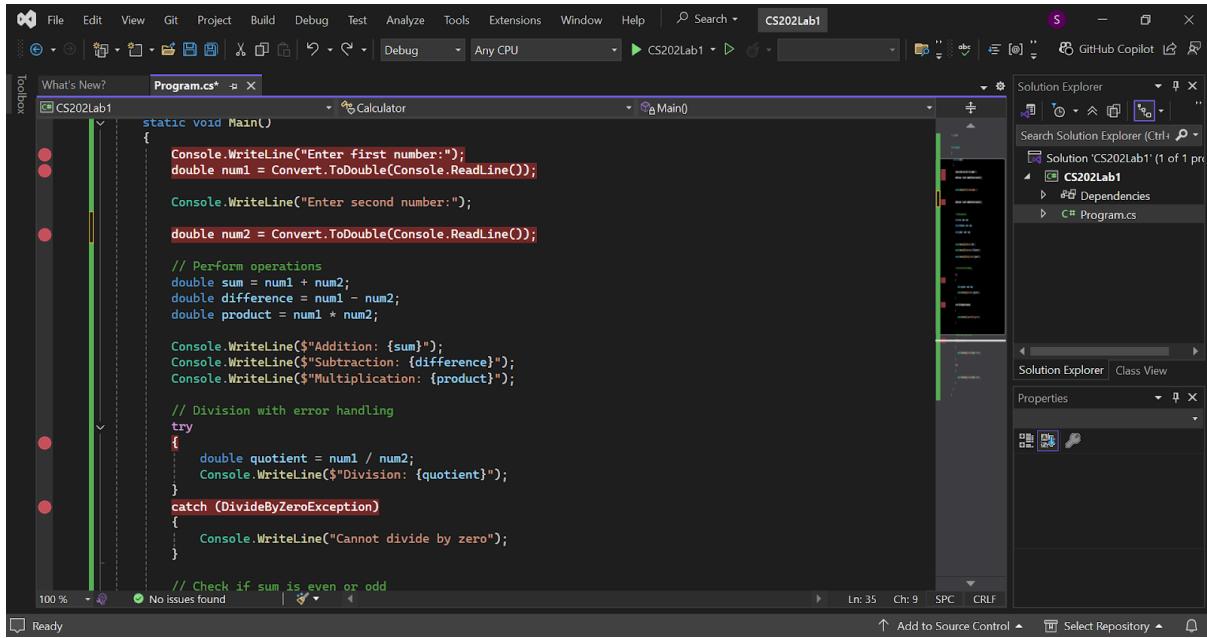
Results for 2 and 1:
Addition: 3
Subtraction: 1
Multiplication: 2
Division: 2
The sum 3 is odd

Enter first number (or 'q' to quit):
q

C:\Users\sawan\source\repos\CS202Lab1\CS202Lab1\bin\Debug\net8.0\CS202Lab1.exe (process 18048) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

6. Debugging using Visual Studio Debugger

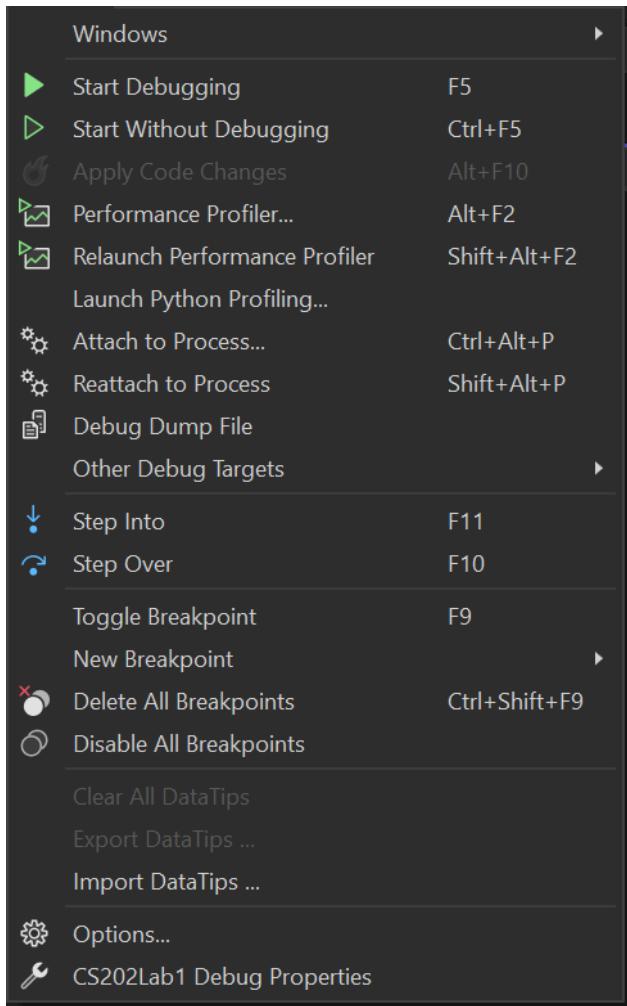
Marking breakpoints in the program



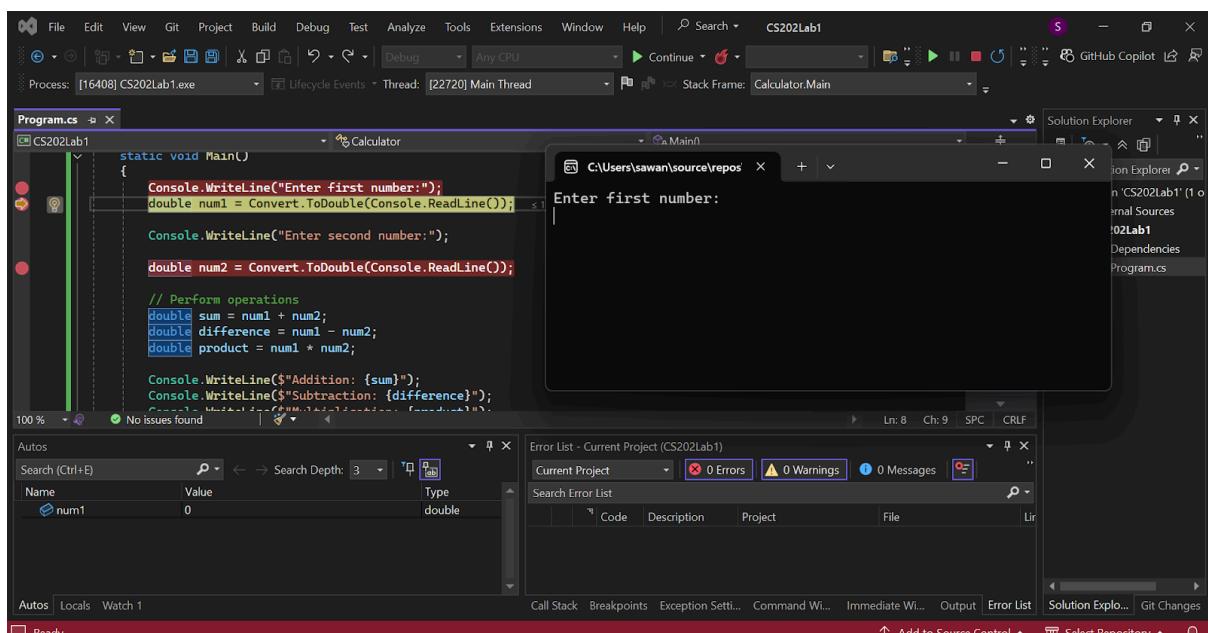
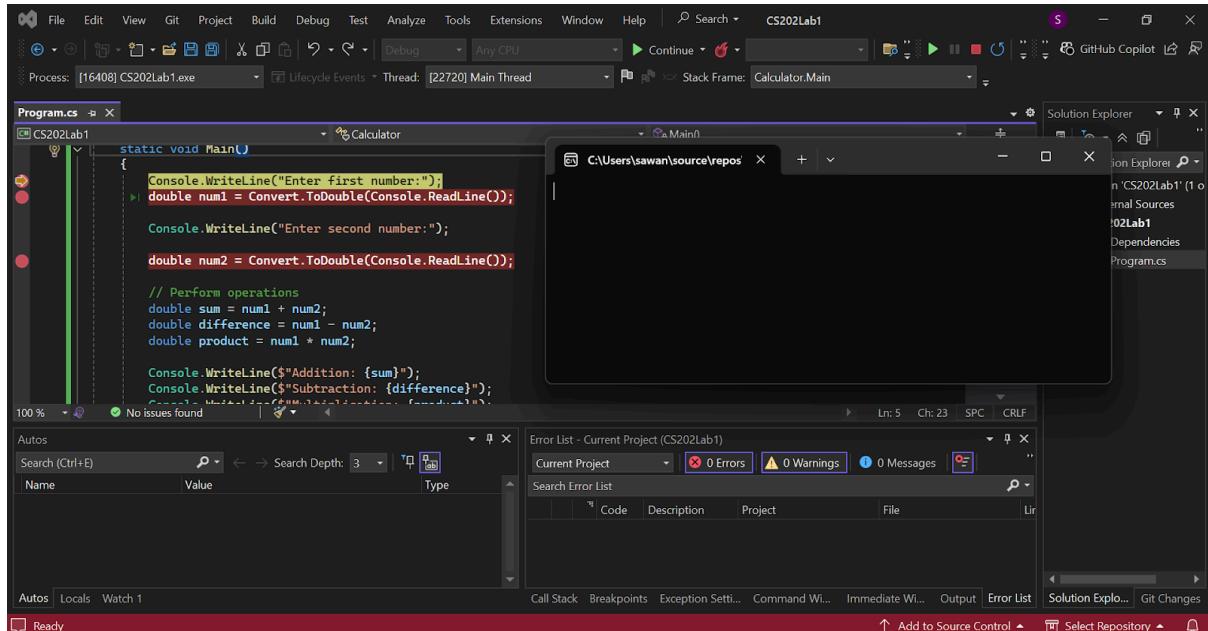
The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'CS202Lab1' with its projects and files.
- Properties View:** Visible on the right side of the interface.
- Code Editor:** Displays the 'Program.cs' file with several red circular breakpoints marked along the left margin.
- Status Bar:** Shows 'No issues found' and other status information like 'Ln: 35 Ch: 9'.

After this go to debug and choose start debugging



Yellow arrows have appeared and program have stop before the breakpoint thus nothing in the terminal and on pressing continue it executes



Giving input but nothing on screen because the input is not being read because the Command not executed.

A screenshot of the Visual Studio IDE showing a debugger session. The main window displays the code for Program.cs, which includes operations like addition, subtraction, multiplication, and division. A break point is set at the division operation. The output window shows the user input "10". The Error List shows no errors or warnings. The Solution Explorer shows the project structure.

```
double num2 = Convert.ToDouble(Console.ReadLine());  
// Perform operations  
double sum = num1 + num2;  
double difference = num1 - num2;  
double product = num1 * num2;  
  
Console.WriteLine($"Addition: {sum}");  
Console.WriteLine($"Subtraction: {difference}");  
Console.WriteLine($"Multiplication: {product}");  
  
// Division with error handling  
try  
{  
    double quotient = num1 / num2;  
}  
catch (Exception ex)  
{  
    Console.WriteLine(ex.Message);  
}
```

Now on execution of the command input appears

A screenshot of the Visual Studio IDE showing a debugger session. The code has been modified to prompt for two numbers. The user has entered "10" for the first number and "0" for the second number. The division operation is currently being executed, as indicated by the status bar message "28.324ms elapsed". The Error List shows no errors or warnings. The Solution Explorer shows the project structure.

```
static void Main()  
{  
    Console.WriteLine("Enter first number:");  
    double num1 = Convert.ToDouble(Console.ReadLine());  
  
    Console.WriteLine("Enter second number:");  
  
    double num2 = Convert.ToDouble(Console.ReadLine()); // 28.324ms elapsed  
  
    // Perform operations  
    double sum = num1 + num2;  
    double difference = num1 - num2;  
    double product = num1 * num2;  
  
    Console.WriteLine($"Addition: {sum}");  
    Console.WriteLine($"Subtraction: {difference}");  
    Console.WriteLine($"Multiplication: {product}");  
}
```

The screenshot shows two instances of a calculator application running in Visual Studio. In the top instance, a breakpoint is set on the division line, and the output window shows:

```
Enter first number:
10
Enter second number:
20
Addition: 30
Subtraction: -10
Multiplication: 200
Division: 0.5
```

In the bottom instance, breakpoints are set on both the division and modulus lines, and the output window shows:

```
Enter first number:
10
Enter second number:
20
Addition: 30
Subtraction: -10
Multiplication: 200
Division: 0.5
```

That's how the program runs on multiple breakpoints and every program have breakpoints like this.

Breakpoints allow you to pause program execution at specific lines of code to inspect variables, control flow, and identify bugs. In Visual Studio:

Setting Breakpoints

Click the left margin (gray area) next to a line number (🔴 red circle appears).

When the program runs, it stops at the breakpoint (yellow arrow ➤).

How It Works

The debugger freezes execution at each breakpoint.

You can inspect variables, modify values, and step through code.

Useful for checking:

Input values (num1, num2)

Arithmetic results (sum, product)

Exception handling (DivideByZeroException)

Breakpoints in Activity 2

BP1: Program start (before first input)

BP2: After num1 is entered

BP3: After num2 is entered

BP4: Before division (checks num2 == 0)

BP5: If division by zero occurs

BP6: Before checking if sum is even/odds

Stepping Commands in Debugging

Step Over: Executes the current line and moves to the next without entering method calls.

F10 Use for simple statements (e.g., Console.WriteLine(), arithmetic operations).

Step Into: Enters a method call to debug inside it (e.g., Convert.ToDouble()).

F11 Use when you want to inspect a method's internal logic.

Step Out: Exits the current method and returns to the caller.

Shift+F11 Use when you're done debugging inside a method.

Example Debugging Session

1. Test Case: num1 = 10, num2 = 0 (division by zero)
2. Program starts, pauses at BP1 (before input).
3. Press F10 (Step Over) to move to num1 input.
4. Enter 10, hits BP2 (after num1 is stored).
5. Hover over num1 → should show 10.
6. Enter 0, hits BP3 (after num2 is stored).

7. Press F10 to move to calculations.
8. Hits BP4 (before division).
9. Press F10 to attempt division.
10. Exception occurs, jumps to BP5 (catch block).
11. Inspect error in Locals Window.
12. Press F10 to see "Cannot divide by zero" message.
13. Hits BP6 (even/odd check).
14. Press F10 to see output ("The sum 10 is even").