CS202 - Lab Assignment 9

- Hitesh Kumar
- 22110098
- https://github.com/Hit2737/CS202_A3

Module Dependency Analysis using Pydeps

Setup for Pydeps

- **Install pydeps in a virtual environment:**

  ```
  $ python3 -m venv venv; source venv/bin/activate
  ```

  ```
  $ pip install pydeps
  ```

Repository Selection

- **Language & Application:** The repository must be primarily Python and represent a real-world application.
- **Popularity Metrics:** A minimum of 10,000+ GitHub stars and 1000+ forks.
- **Community Engagement:** The repositories should have at least 200 active contributors and recent commit activity (e.g., within the last 6 months) and a minimum of 10,000 commits.
- **Modularized:** The repository should consist of different modules with dependencies on each other, which will be used in the analysis from pydeps.
- **Tool Assistance:** The SEART GitHub Search Engine filters projects based on these criteria.

Based on the above criteria, the selected repository selected using SEART is:

Run Pydeps on Selected Repository

1. **Clone the nltk repository:**

   ```
   $ git clone https://github.com/nltk/nltk.git
   ```

2. **Run pydeps on different modules (as there is no main file which runs everything)**

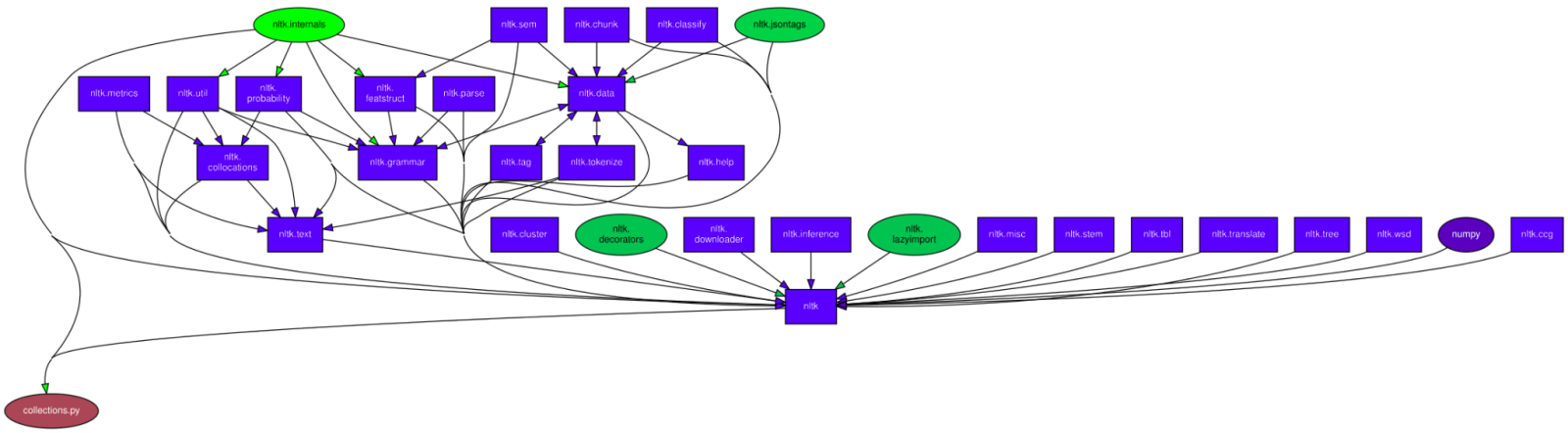
   ```
   $ pydeps nltk/nltk/collections.py --noshow -o
   ./results/nltk_collections.svg
   ```

   ```
   $ pydeps nltk/nltk/books.py --noshow -o ./results/nltk_books.svg
   ```

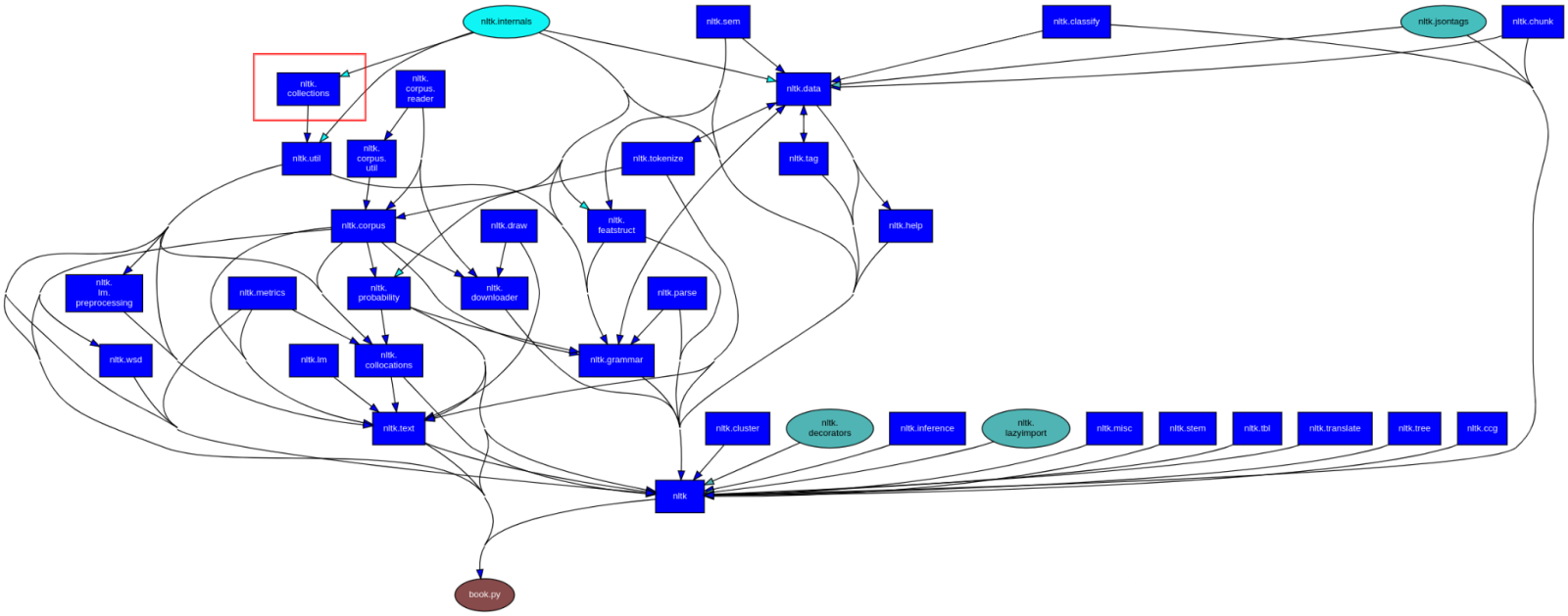
   You may run on many more modules for a complete analysis of how every module depends on others we will boil down our focus on the `collection.py` module.

3. **To also generate the JSON files for further analysis, we can use the following command**

   ```
   $ pydeps nltk/nltk/collections.py --show-deps --noshow -o
   ./results/nltk_collections.svg > ./results/nltk_collections.json
   ```



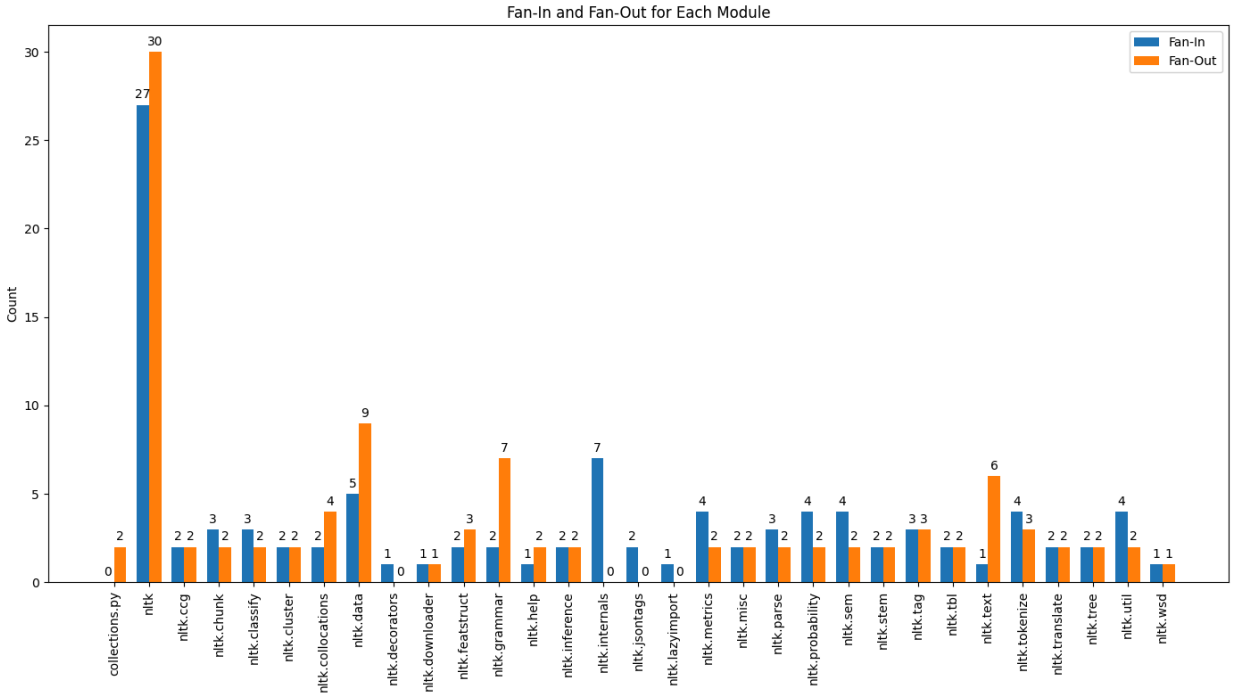
Run on collections.py



collections.py used in books.py

Analysis of the Pydeps Output (collections.py)

The generated JSON file can further analyze the module dependencies in the repository.



Calculating fan-in and fan-out:

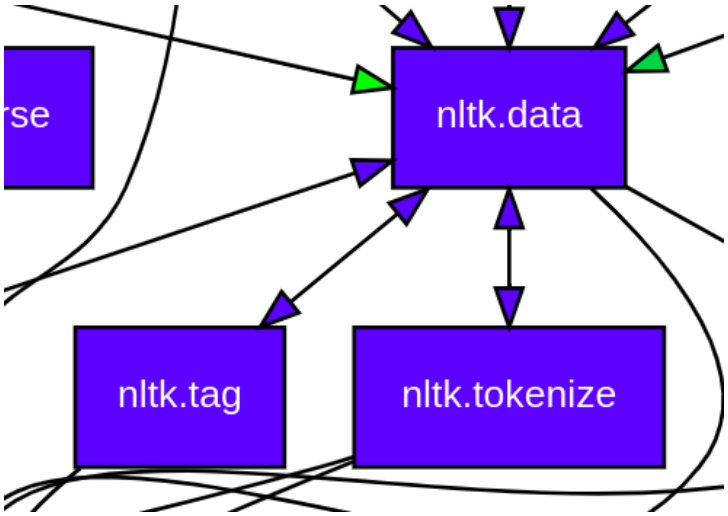
A simple function does this job. It analyzes the JSON file to calculate how many times others have imported a particular module and how many modules it imports.

Highly Coupled Modules

- **nltk** is a central hub that imports and is being imported by many modules (e.g., nltk.chunk, nltk.ccg).
- **nltk.data** and **nltk.internals** also show high coupling, which is widely shared throughout the code.

Cyclic Dependencies

- Cycles exist, such as **nltk.data** ⟷ **nltk.tokenize**, **nltk.data** ⟷ **nltk.tag** complicating refactoring and testing.
- Cycles force changes in one module to affect the other, reducing modularity.



Unused/Disconnected Modules

- All the modules are used by one another, but some modules might not be used by other modules and are standalone. For that, we have to run our analysis on all the different modules in the repo or

- Most other modules are interconnected, so there is no truly unused module.

Depth of Dependencies

- The architecture is relatively shallow but with many lateral references.
- The central “nltk” module ties everything together, making it a single-layer web rather than a layered hierarchy.

Brief Impact Assessment

- **Core Module Changes:**
  Since `nltk` is central and widely used, any change—even minor—in its functionality or API can cause a ripple effect throughout the system, leading to unexpected failures in dependent modules.

- **At-Risk Modules:**
  Modules that directly import `nltk` or are part of cyclic dependencies are most vulnerable. These tightly coupled components are highly sensitive to any modifications in `nltk` and are likely to break if their behaviour changes unexpectedly.

Measuring Java Class Cohesion using LCOM

Choosing Java Project

The **JAVA** project must contain at least 10 classes, and the selection criteria are used as mentioned above.

The selected repository is **00-Evan/shattered-pixel-dungeon,** a repository for the game Shattered Pixel Dungeon (available on Play Store).

Running LCOM on the Project

Assuming that the **LCOM.jar** and the project folder are in the same directory.

Run the following command in the terminal to generate the LCOM metrics using the LCOM.jar.

```
$ java -jar LCOM.jar -i <project_folder_path> -o <output_folder>
```

As shown in the figure below, a **CSV** will be generated in the **<output_folder>** containing the LCOM1 to LCOM5 and YALCOM values.

	A	B	C	D	E	F	G	H	I
1	**Project Name**	**Package Name**	**Type Name**	**LCOM1**	**LCOM2**	**LCOM3**	**LCOM4**	**LCOM5**	**YALCOM**
2	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	PylonSprite	20.0	19.0	6.0	4.0	0.8333333333333334	0.5714285714285714
3	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	MonkSprite	0.0	0.0	1.0	1.0	-0.0	0.0
4	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	GnollSapperSprite	1.0	0.0	2.0	2.0	0.0	1.0
5	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	BruteSprite	0.0	0.0	1.0	1.0	0.0	0.0
6	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	SuccubusSprite	1.0	0.0	2.0	2.0	0.0	1.0
7	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	SheepSprite	0.0	0.0	1.0	1.0	0.0	0.0
8	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	GnollGeomancerSprite	39.0	23.0	4.0	1.0	0.65	0.0
9	shattered-pixel-dungeon	com.shatteredpixel.shatteredpixeldungeon.sprites	UndeadSprite	3.0	0.0	3.0	3.0	0.0	1.0

After analyzing the output **CSV,** it was found that many classes have High LCOM values, but the **GAMESCENE** class contains the highest LCOM values among all the classes.

SupportButton	1.0	0.0	2.0	2.0	0.0	1.0
GameScene	3897.0	3699.0	37.0	14.0	0.9658585858585857	0.07692307692307...
NewsScene	62.0	46.0	5.0	5.0	0.8645833333333334	0.38461538461538...

The main reason is that **GameScene** manages the main game scene in Shattered Pixel Dungeon, and its responsibilities include:

- **Rendering & Visuals:** Drawing water, tiles, fog, and other effects.
- **UI Management:** Handling menus, toolbars, status panes, and pop-ups.
- **Game Logic:** Coordinating the actor thread and processing game events.
- **Utility Operations:** Managing sounds, animations, and state transitions.

The GameScene class consists of approximately 75 methods with similar and different functionalities, including create(), update(), destroy(), layouts (), and gameOver() - showing similar functionalities related to the game

initiation and completion, addcustomWalls(), updateAvatar(), resetMap() - among others showing different functionalities which can be refactored into different classes to reduce cohesion.

What High LCOM Values Suggest

- **Low Cohesion:** Very high LCOM values (e.g., LCOM1=3897.0, LCOM2=3699.0) indicate that many methods operate on unrelated fields. The lower LCOM3/LCOM4 values and ratios indicate some subgroupings are slightly more cohesive, but overall cohesion is very low.
- **Multiple Responsibilities:** The class acts as a "God Class" by handling diverse and loosely related tasks.
- **Refactoring Opportunity:** The class is a prime candidate for functional decomposition to improve maintainability.

Functional Decomposition Ideas

- **Separate Rendering:** Extract tile, water, fog, and animation logic into dedicated rendering classes.
- **Isolate UI Management:** Move menus, toolbars, and status panes into their own classes.
- **Delegate Game Logic:** Offload actor processing and event handling into separate controllers.

Refactoring the class into smaller, focused components will improve cohesion, enhance maintainability, and simplify testing.

Visualizing Cohesion in Some Classes

Class Name	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM
TitleScene	209.0	208.0	20.0	19.0	0.9833333	0.904761904

NewsButton	3.0	3.0	3.0	3.0	1.0	1.0
GnollGeomancerSprite	39.0	23.0	4.0	1.0	0.65	0.0
MonkSprite	0.0	0.0	1.0	1.0	-0.0	0.0
Dungeon	999.0	963.0	38.0	19.0	0.93436692	0.0434782608
GameScene	3897.0	3699.0	37.0	14.0	0.96585858	0.0769230769

Comparing the Classes based on Different Cohesion Values

- **NewsButton:** LCOM values are very low (3) with a perfect YALCOM (1.0) – a highly cohesive, single-purpose component.
- **TitleScene:** Moderate LCOM (~209) and a high YALCOM (0.9048) – despite handling multiple tasks, its methods reasonably share standard functionality.
- **GnollGeomancerSprite:** Moderate LCOM (39/23) but a YALCOM of 0.0 – indicating little to no cohesion among its methods.
- **MonkSprite:** Minimal LCOM values (0–1) and YALCOM of 0.0 – suggesting a trivial class with almost no shared functionality.
- **Dungeon:** High LCOM (999/963) with a very low YALCOM (0.0435) – a class with many unrelated responsibilities, reflecting low cohesion.
- **GameScene:** Extremely high LCOM (3897/3699) and a very low YALCOM (0.0769) – exhibit low cohesion due to handling a wide range of unrelated tasks.

CS202 - Lab Assignment 10

- Hitesh Kumar
- 22110098

Prerequisites

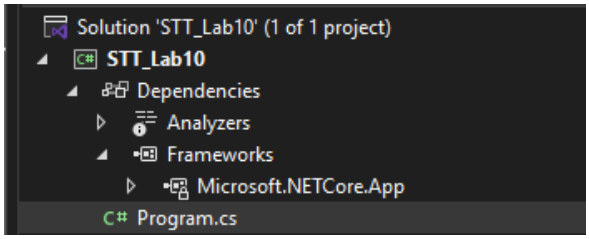
- **Operating System:** Windows
- **Software:** Visual Studio 2022 (Community Edition), Visual Studio with .NET SDK
- **Programming Language:** C# (latest stable version)

Activities

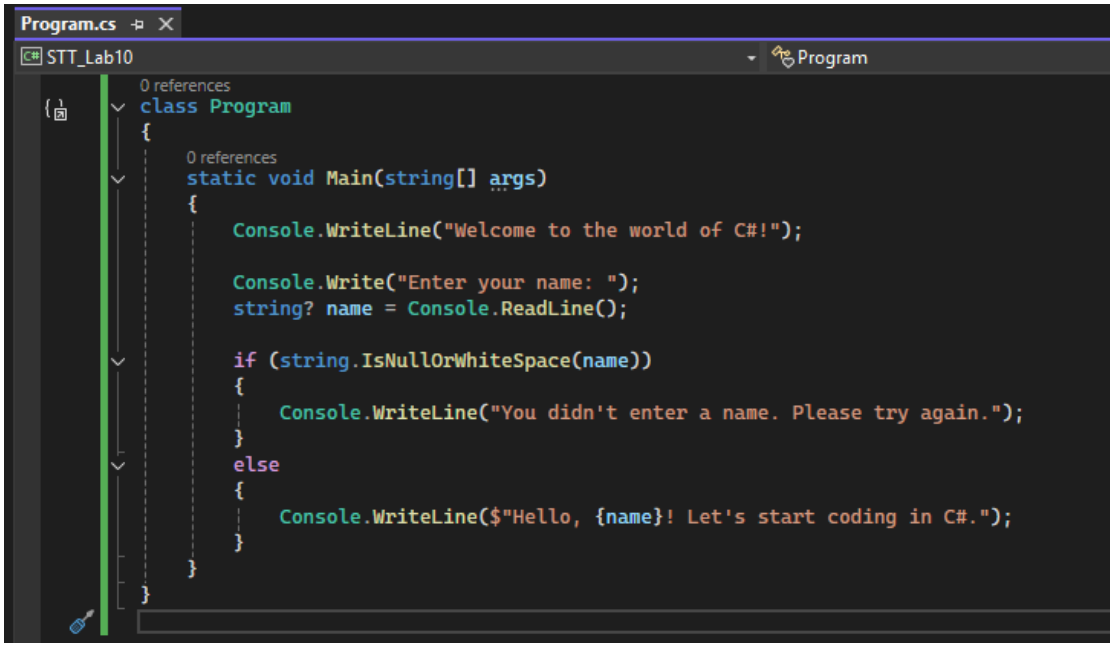
Activity 1: Setting Up .NET Development Environment

Task Description:

- Open Visual Studio and create a new C# Console Application project.
- Ensure that the target framework is set to **.NET 6.0** or later.



- Write and execute a simple C# program that prints a welcome message.

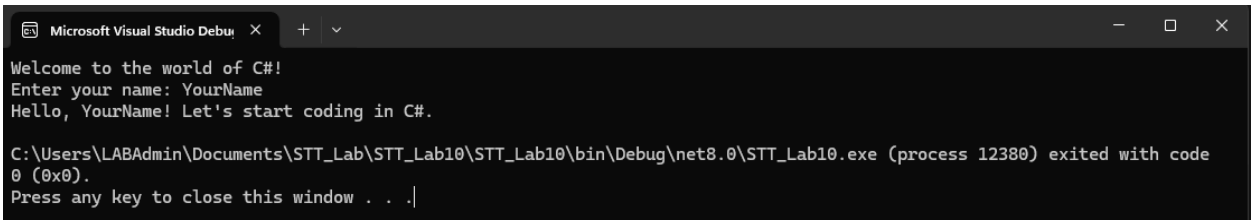


Objective:

- Set up the Visual Studio development environment.
- Verify successful execution of the project.

Expected Output:

The program displays a welcome message, confirming that the environment is configured correctly.



Activity 2: Understanding Basic Syntax and Control Structures

Task Description:

- Create a `Calculator` class to perform basic arithmetic operations.

```
public class Calculator
{
    0 references
    public double a { get; set; }
    0 references
    public double b { get; set; }
    1 reference
    public double Add(double a, double b){
        return a + b;
    }
    1 reference
    public double Subtract(double a, double b){
        return a - b;
    }
    1 reference
    public double Multiply(double a, double b){
        return a * b;
    }
    1 reference
    public double Divide(double a, double b){
        if (b == 0){
            Console.WriteLine("Cannot divide by zero.");
            return double.NaN;
        }
        return a / b;
    }
    1 reference
    public void CheckEvenOrOdd(double num){
        if (num % 2 == 0){
            Console.WriteLine($"{num} is an even number.");
        }else{
            Console.WriteLine($"{num} is an odd number.");
        }
    }
}
```

- Accept two numbers from the user as input.
- Use an if-else condition to determine whether the sum is even or odd.
- Display the results using `Console.WriteLine()`.

```
class Program
{
    0 references
    static void Main()
    {
        Calculator calc = new Calculator();

        Console.Write("Enter the first number: ");
        double num1 = Convert.ToDouble(Console.ReadLine());

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

        double sum = calc.Add(num1, num2);
        double difference = calc.Subtract(num1, num2);
        double product = calc.Multiply(num1, num2);
        double quotient = calc.Divide(num1, num2);

        Console.WriteLine("\n---- Results ----");
        Console.WriteLine($"Addition: {num1} + {num2} = {sum}");
        Console.WriteLine($"Subtraction: {num1} - {num2} = {difference}");
        Console.WriteLine($"Multiplication: {num1} * {num2} = {product}");

        if (!double.IsNaN(quotient))
        {
            Console.WriteLine($"Division: {num1} / {num2} = {quotient:F2}");
        }

        calc.CheckEvenOrOdd(sum);

        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }
}
```

Objective:

- Understand basic input/output operations in C#.
- Explore if-else conditions and arithmetic operations.

Expected Output:

- Results of all arithmetic operations.
- A message indicating whether the sum is even or odd.

```
C:\Users\LABAdmin\Documer
Enter the first number: 48
Enter the second number: 24

---- Results ----
Addition: 48 + 24 = 72
Subtraction: 48 - 24 = 24
Multiplication: 48 * 24 = 1152
Division: 48 / 24 = 2.00
72 is an even number.
```

Activity 3: Implementing Loops and Functions

Task Description:

- Used a **for loop** to print numbers from 1 to 10.
  Used a **while loop** to repeatedly ask for user input until they type `exit`.
- Defined a function to calculate the factorial of a given number and display the result.

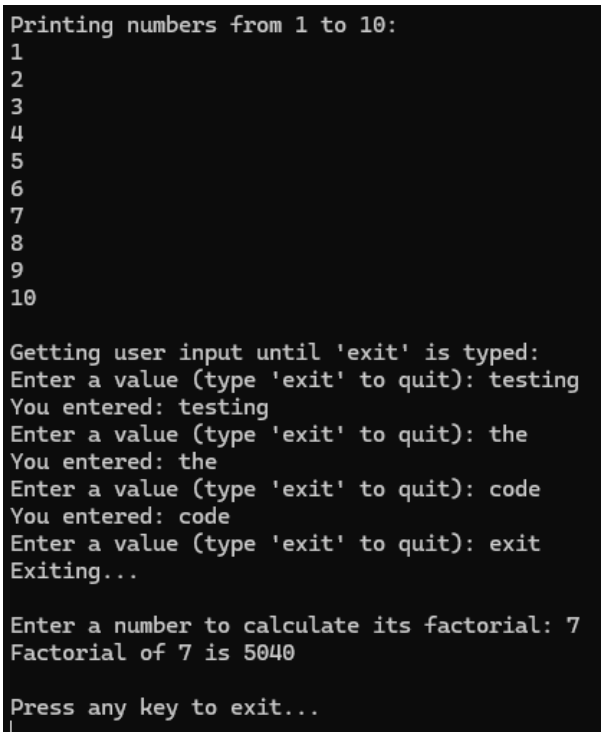
```
public void PrintNumbers()
{
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine(i);
    }
}

1 reference
public void GetUserInput()
{
    string? input;
    while (true)
    {
        Console.Write("Enter a value (type 'exit' to quit): ");
        input = Console.ReadLine()?.ToLower();
        if (input == "exit")
        {
            Console.WriteLine("Exiting...");
            break;
        }
        Console.WriteLine($"You entered: {input}");
    }
}

1 reference
public int CalculateFactorial(int num)
{
    int fact = 1;
    for (int i = 1; i <= num; i++)
    {
        fact *= i;
    }
    return fact;
}
```

Objective:

- Explore different looping mechanisms in C#.
- Learn to define and call functions in a C# program.

Expected Output:

- Numbers from 1 to 10.
- The program keeps asking for user input until `exit` is typed.
- The factorial of the provided number is displayed.



Activity 4: Object-Oriented Programming in C#

Task Description:

- Create a `Student` class with properties `Name`, `ID`, and `Marks`.
- Implement a method `getGrade()` that returns grades (`A`, `B`, `C`, etc.) based on the marks.

```
5 references
public class Student
{
    2 references
    public string Name { get; set; }
    2 references
    public string ID { get; set; }
    6 references
    public double Marks { get; set; }

    2 references
    public Student(string name, string id, double marks)
    {
        Name = name;
        ID = id;
        Marks = marks;
    }

    1 reference
    public string 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";
    }

    2 references
    public void DisplayDetails()
    {
        Console.WriteLine($"Name: {Name}");
        Console.WriteLine($"ID: {ID}");
        Console.WriteLine($"Marks: {Marks}");
        Console.WriteLine($"Grade: {GetGrade()}");
    }
}
```

- Create a subclass `StudentIITGN` that inherits from `Student` and adds a new property `Hostel_Name_IITGN`.

```
3 references
public class StudentIITGN : Student
{
    2 references
    public string Hostel_Name_IITGN { get; set; }

    1 reference
    public StudentIITGN(string name, string id, double marks, string hostelName)
        : base(name, id, marks)
    {
        Hostel_Name_IITGN = hostelName;
    }

    1 reference
    public new void DisplayDetails()Z
    {
        base.DisplayDetails();
        Console.WriteLine($"Hostel Name: {Hostel_Name_IITGN}");
    }
}
```

- Use the `Main()` method to create and display IITGN student details.

```
Student student = new Student(name, id, marks);
Console.WriteLine("\n---- Student Details ----");
student.DisplayDetails();

Console.WriteLine("\nEnter IITGN student details:");

Console.Write("Enter Hostel Name: ");
string? hostelName = Console.ReadLine();
if (string.IsNullOrEmpty(hostelName))
{
    Console.WriteLine("Hostel Name cannot be empty.");
    return;
}

StudentIITGN iitgnStudent = new StudentIITGN(name, id, marks, hostelName);
Console.WriteLine("\n---- IITGN Student Details ----");
iitgnStudent.DisplayDetails();
```

Expected Output:

- Student details, including name, ID, marks, grade, and hostel name, are displayed.

```
Enter student details:
Enter Name: Hitesh
Enter ID: 1234
Enter Marks: 98

---- Student Details ----
Name: Hitesh
ID: 1234
Marks: 98
Grade: A

Enter IITGN student details:
Enter Hostel Name: Emiet

---- IITGN Student Details ----
Name: Hitesh
ID: 1234
Marks: 98
Grade: A
Hostel Name: Emiet

Press any key to exit...
```

Activity 5: Exception Handling

Task Description:

- Modify the program from Activity 2 to include exception handling.
- Use a try-catch block to handle **division-by-zero** errors.

```
1 reference
public double Divide(double a, double b)
{
    try
    {
        if (b == 0)
        {
            throw new DivideByZeroException("Cannot divide by zero.");
        }
        return a / b;
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine(ex.Message);
        return double.NaN;
    }
}
```

- Ensure that invalid input does not crash the program by catching format exceptions

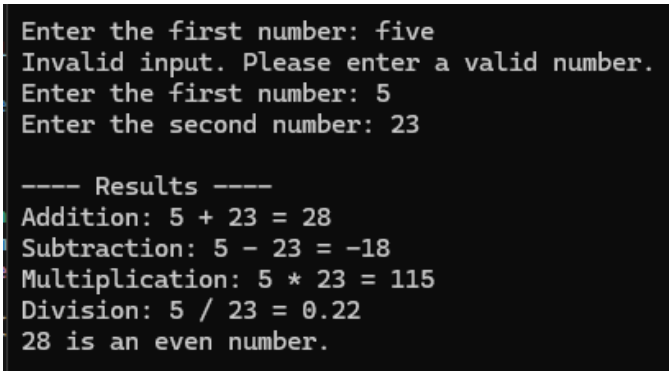
```
while (true)
{
    try
    {
        Console.Write("Enter the first number: ");
        num1 = Convert.ToDouble(Console.ReadLine());
        break;
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid input. Please enter a valid number.");
    }
}

while (true)
{
    try
    {
        Console.Write("Enter the second number: ");
        num2 = Convert.ToDouble(Console.ReadLine());
        break;
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid input. Please enter a valid number.");
    }
}
```

Objective:

- Learn to handle exceptions gracefully in C#.
- Prevent program crashes due to invalid user input or runtime errors.

Expected Output:

- Appropriate error messages for division-by-zero and invalid input scenarios.
- The program continues execution even after handling exceptions.

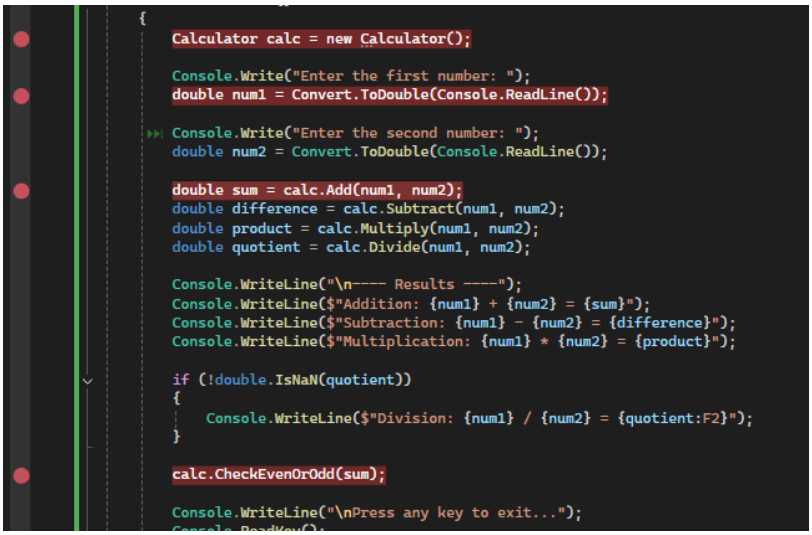


Activity 6: Debugging Using Visual Studio Debugger

Task Description:

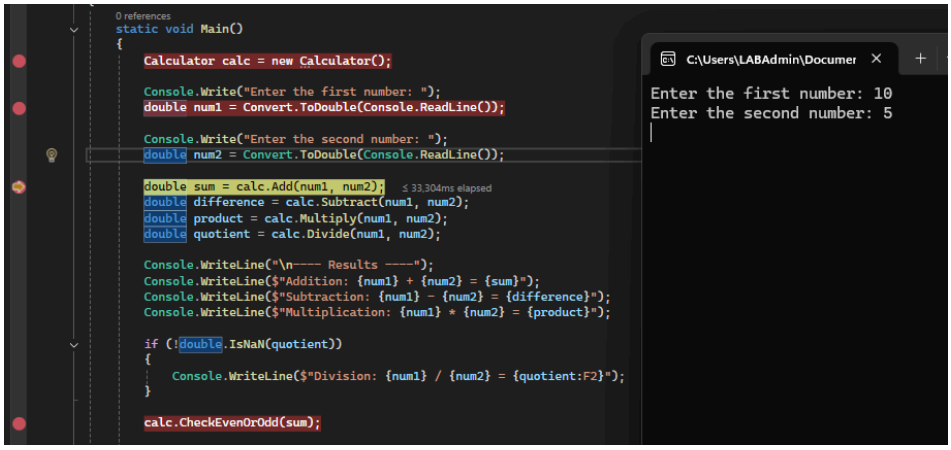
- Set breakpoints in the code to pause execution at key points.
- Use **Step-In (F11)** to step into method calls and observe execution line by line.
- Use **Step-Over (F10)** to execute methods without entering them.
- Use **Step-Out (Shift + F11)** to exit from the current method and return to the caller.

Activity 2:



```
{
    Calculator calc = new Calculator();

    Console.Write("Enter the first number: ");
    double num1 = Convert.ToDouble(Console.ReadLine());

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

    double sum = calc.Add(num1, num2);
    double difference = calc.Subtract(num1, num2);
    double product = calc.Multiply(num1, num2);
    double quotient = calc.Divide(num1, num2);

    Console.WriteLine("\n---- Results ----");
    Console.WriteLine($"Addition: {num1} + {num2} = {sum}");
    Console.WriteLine($"Subtraction: {num1} - {num2} = {difference}");
    Console.WriteLine($"Multiplication: {num1} * {num2} = {product}");

    if (!double.IsNaN(quotient))
    {
        Console.WriteLine($"Division: {num1} / {num2} = {quotient:F2}");
    }

    calc.CheckEvenOrOdd(sum);

    Console.WriteLine("\nPress any key to exit...");
```

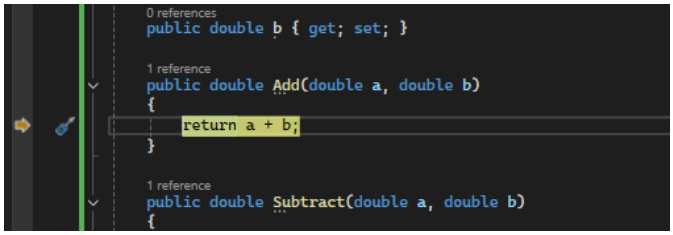
Adding BreakPoints in Code



```
0 references
static void Main()
{
    Calculator calc = new Calculator();

    Console.Write("Enter the first number: ");
    double num1 = Convert.ToDouble(Console.ReadLine());

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

    double sum = calc.Add(num1, num2);   ≤ 33,304ms elapsed
    double difference = calc.Subtract(num1, num2);
    double product = calc.Multiply(num1, num2);
    double quotient = calc.Divide(num1, num2);

    Console.WriteLine("\n---- Results ----");
    Console.WriteLine($"Addition: {num1} + {num2} = {sum}");
    Console.WriteLine($"Subtraction: {num1} - {num2} = {difference}");
    Console.WriteLine($"Multiplication: {num1} * {num2} = {product}");

    if (!double.IsNaN(quotient))
    {
        Console.WriteLine($"Division: {num1} / {num2} = {quotient:F2}");
    }

    calc.CheckEvenOrOdd(sum);
```

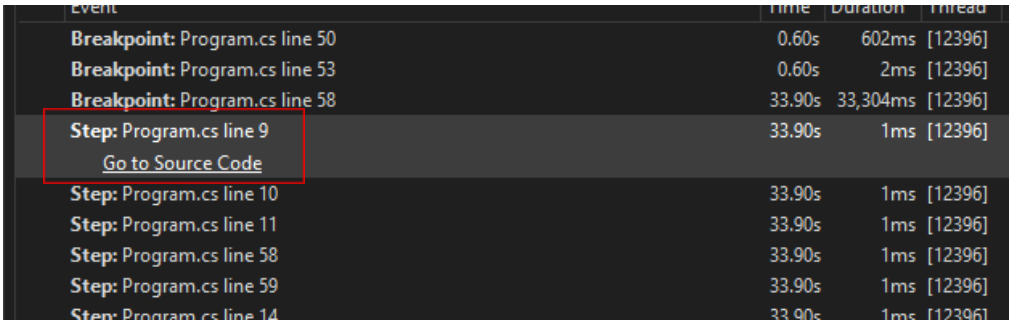
```
C:\Users\LABAdmin\Documer
Enter the first number: 10
Enter the second number: 5
```

Execution pauses at the Breakpoints



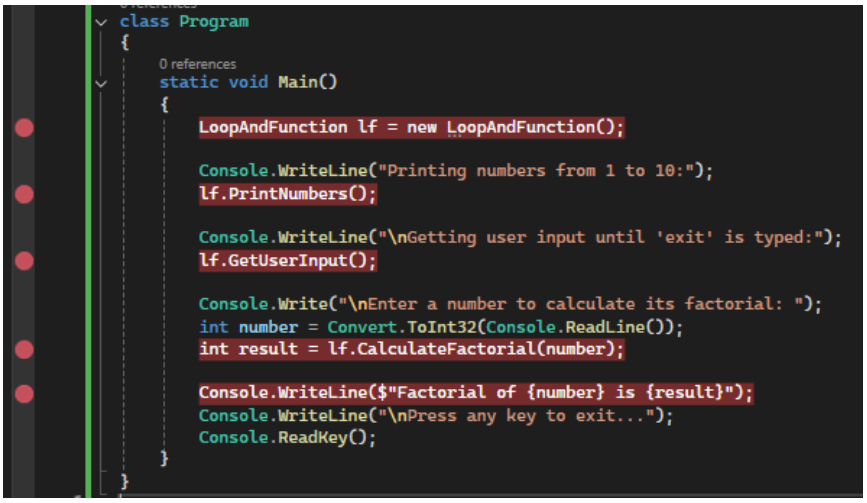
```
0 references
public double b { get; set; }

1 reference
public double Add(double a, double b)
{
    return a + b;
}

1 reference
public double Subtract(double a, double b)
{
```

Use F11 for Step-In to step into methods to execute code line by line. Use Shift F11 to Step-Out for exit the current method and return to the caller shown in the previous image.



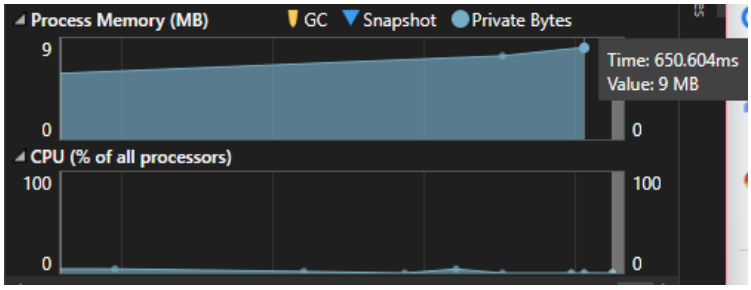
Program jumped into the method at line number 9 using F11

Activity 3:



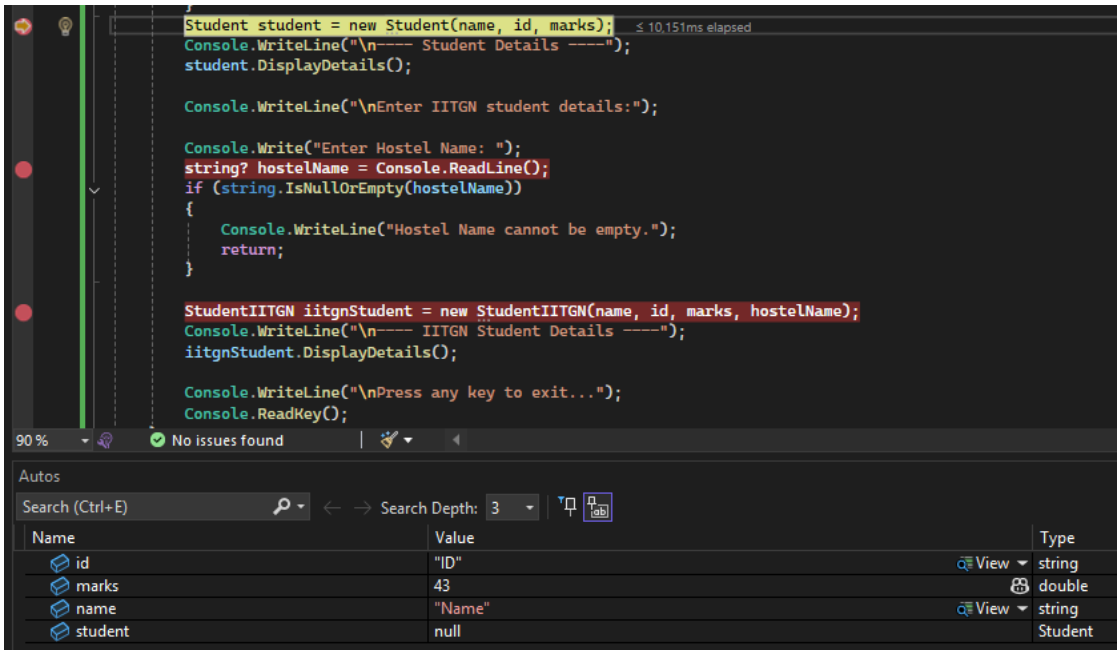
```
class Program
{
    0 references
    static void Main()
    {
        LoopAndFunction lf = new LoopAndFunction();

        Console.WriteLine("Printing numbers from 1 to 10:");
        lf.PrintNumbers();

        Console.WriteLine("\nGetting user input until 'exit' is typed:");
        lf.GetUserInput();

        Console.Write("\nEnter a number to calculate its factorial: ");
        int number = Convert.ToInt32(Console.ReadLine());
        int result = lf.CalculateFactorial(number);

        Console.WriteLine($"Factorial of {number} is {result}");
        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }
}
```

Adding BreakPoints in Code

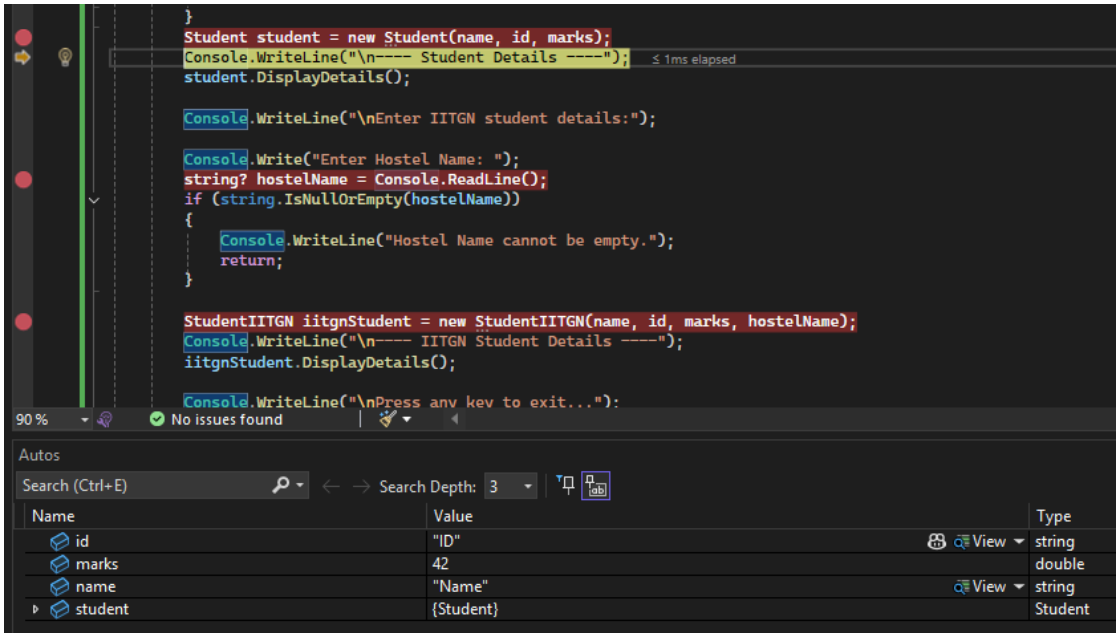


Observe Process Memory and CPU Usage at the time of breakpoint

Activity 4:

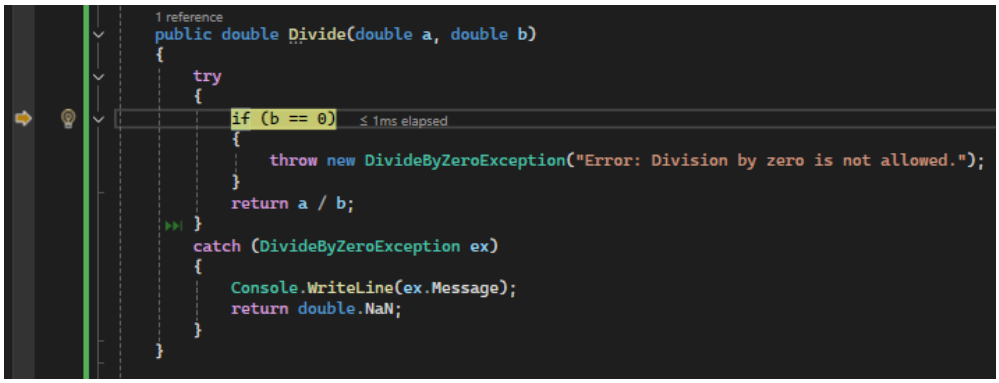


The student class is not being created at the breakpoint because the instruction is paused here



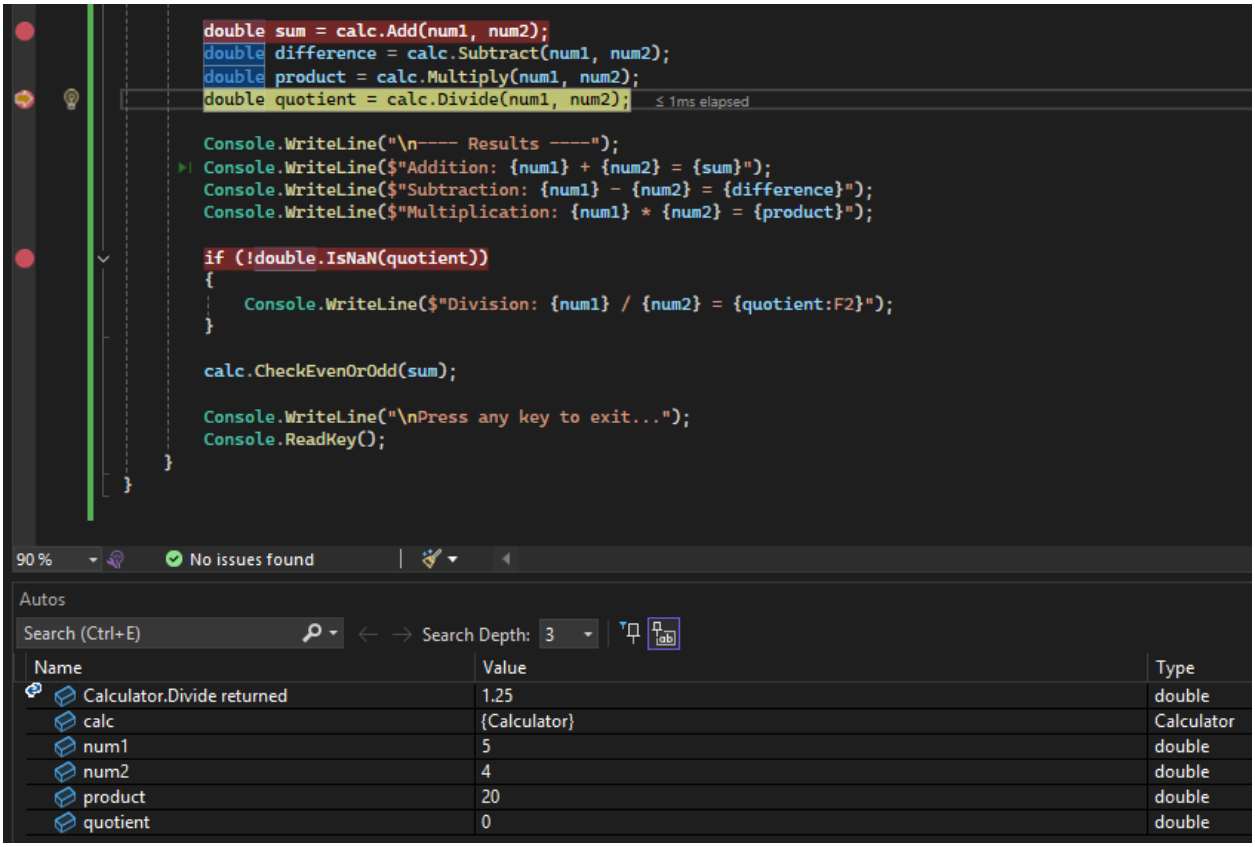
Step-Over takes you to next codeline directly after executing the current line, here also the student class is created after execution of current line

Activity 5:



```
1 reference
public double Divide(double a, double b)
{
    try
    {
        if (b == 0)   ≤ 1ms elapsed
        {
            throw new DivideByZeroException("Error: Division by zero is not allowed.");
        }
        return a / b;
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine(ex.Message);
        return double.NaN;
    }
}
```

Using Step-In, enter into method for line-by-line execution



```
        double sum = calc.Add(num1, num2);
        double difference = calc.Subtract(num1, num2);
        double product = calc.Multiply(num1, num2);
        double quotient = calc.Divide(num1, num2);   ≤ 1ms elapsed

        Console.WriteLine("\n---- Results ----");
        Console.WriteLine($"Addition: {num1} + {num2} = {sum}");
        Console.WriteLine($"Subtraction: {num1} - {num2} = {difference}");
        Console.WriteLine($"Multiplication: {num1} * {num2} = {product}");

        if (!double.IsNaN(quotient))
        {
            Console.WriteLine($"Division: {num1} / {num2} = {quotient:F2}");
        }

        calc.CheckEvenOrOdd(sum);

        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }
}
```

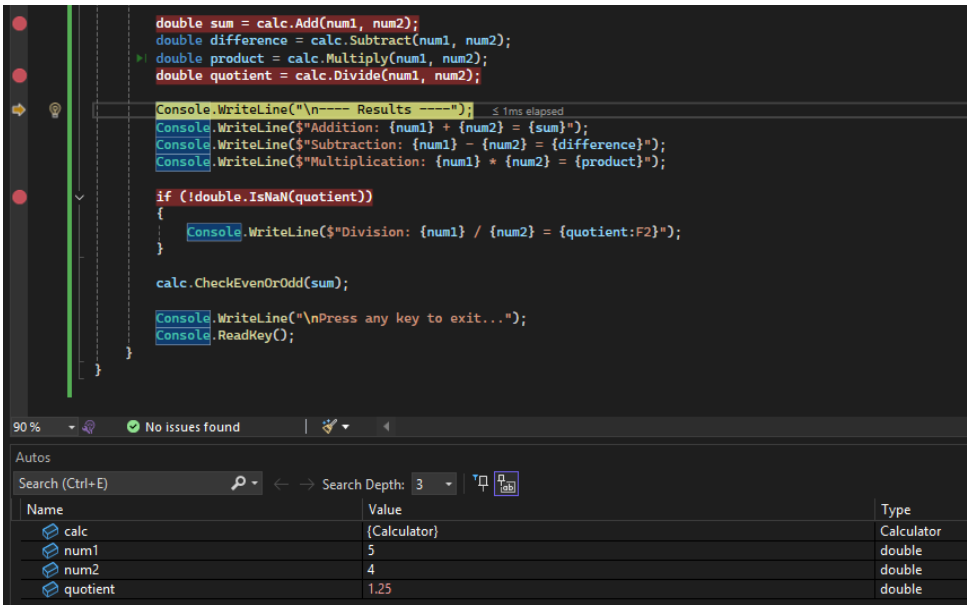
90 % | No issues found

Autos

Search (Ctrl+E) | Search Depth: 3

Name	Value	Type
Calculator.Divide returned	1.25	double
calc	{Calculator}	Calculator
num1	5	double
num2	4	double
product	20	double
quotient	0	double

On Step-Out, return to the caller with quotient not being assigned its value till now



On Step-In again, It takes you to next codeline and the quotient is calculated on successful execution of the instruction

Objective:

- Learn to effectively debug C# programs using Visual Studio Debugger.
- Analyze variable values and track program execution using Step-In, Step-Over and Step-Out.

Conclusion

In this lab, the following tasks were successfully completed:

- Configured the .NET development environment and executed a simple C# program.
- Implemented basic control structures, loops, and functions.
- Applied object-oriented programming concepts using class inheritance and method overriding.
- Handled exceptions effectively and ensured input validation.
  Used the Visual Studio Debugger to analyze program execution and identify errors.
