Eric Gathinji

Programming in C# CST-150-0500

Grand Canyon University

13th July 2025

Activity 5 part 2 & 3

Github link: <https://github.com/Ericgathinji444/Inventory-App-5-1->

Video link: <https://youtu.be/_FCD9Dap5NY?si=x6Dwn_zQc2Pbd7Wt>

FLOWCHART

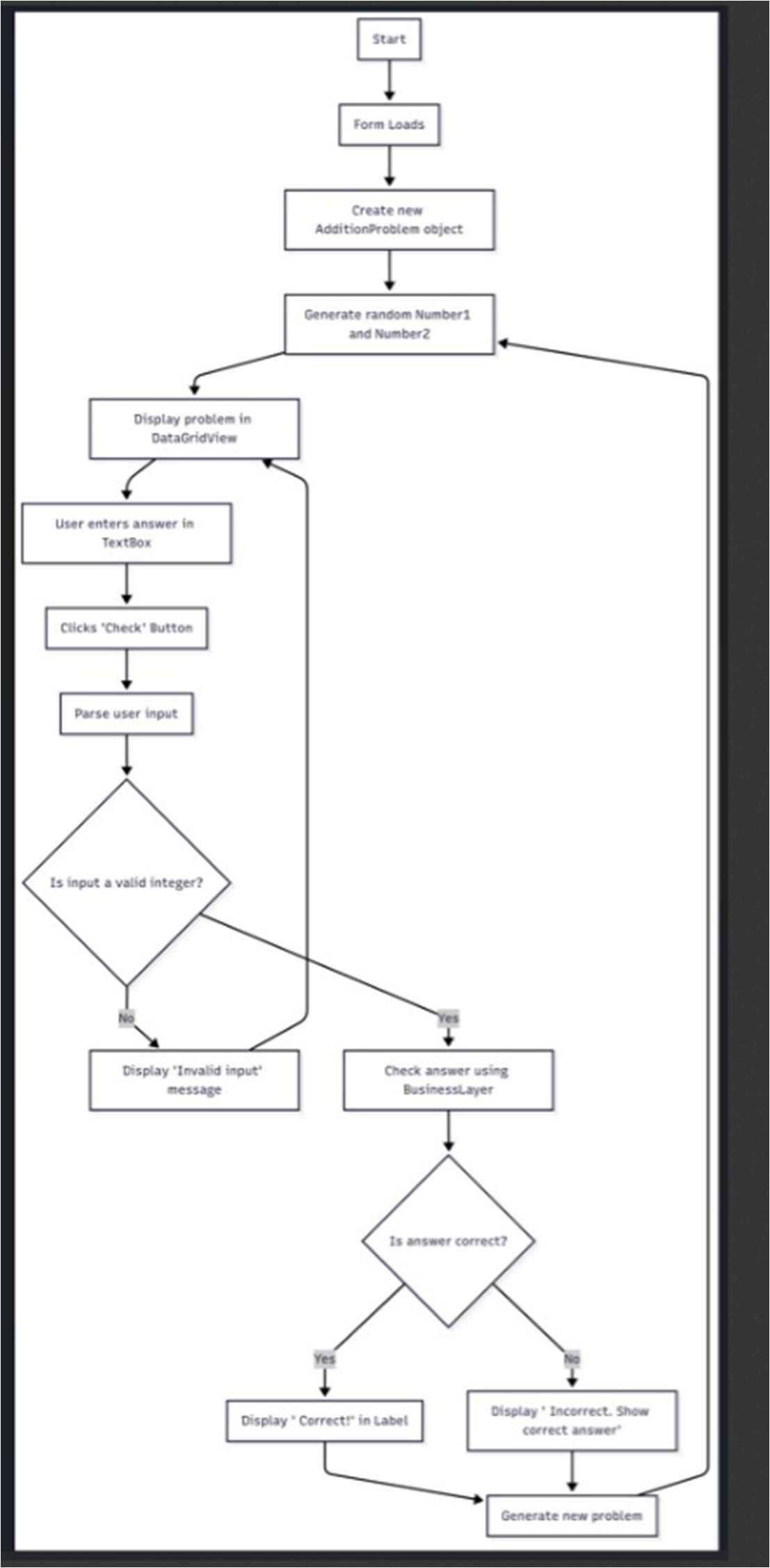


Figure 1:Flowchart showing the control flow of the Addi on Tutor application.

“The flowchart shows how the user interface interacts with the business logic class. On form load, an Addi onProblem object is created and random numbers are generated.

After the user submits an answer, the app checks if the input is valid, then verifies the answer. Feedback is shown accordingly, and a new problem is generated.”

SCREENSHOT OF SOLUTION EXPLORER

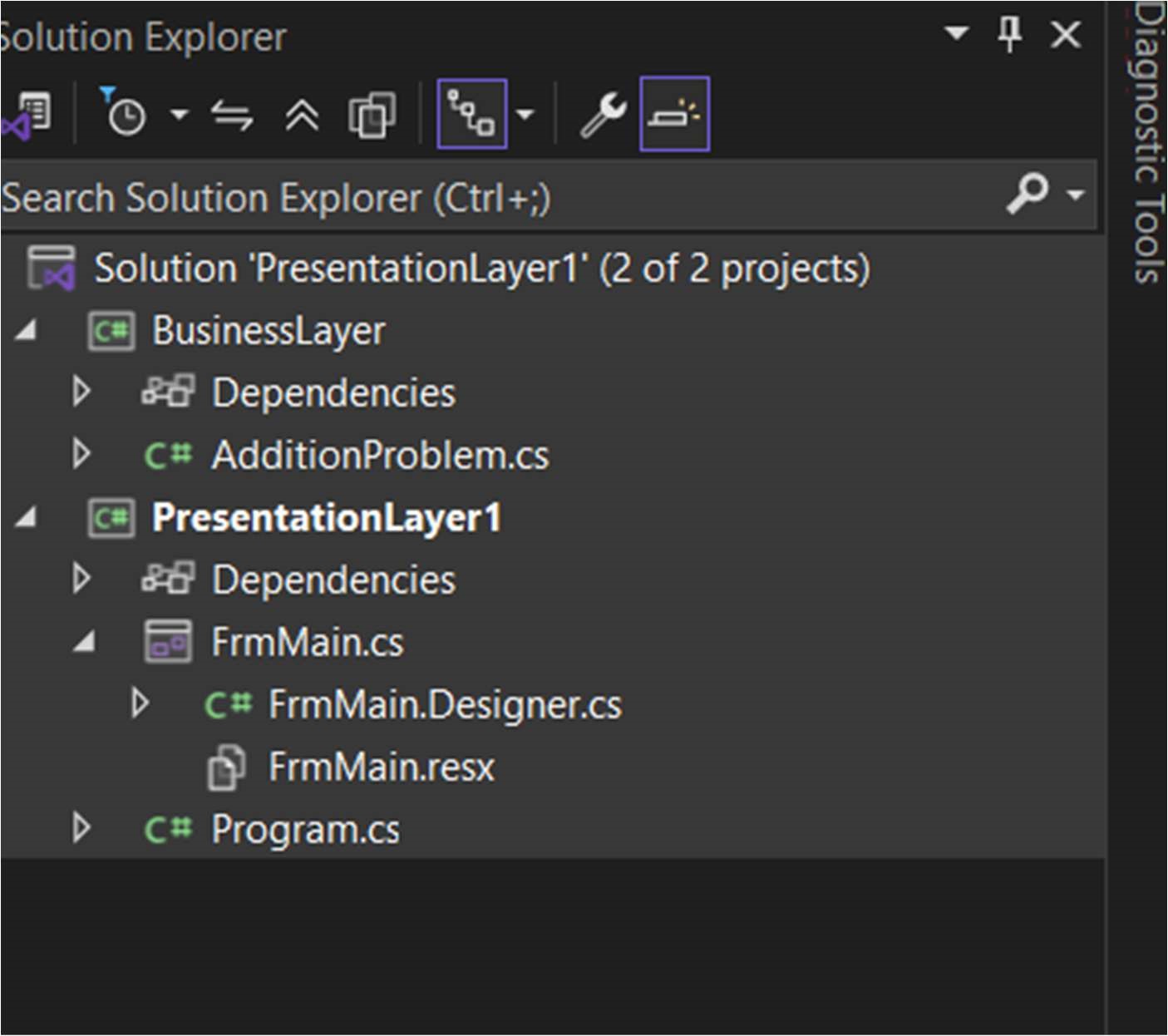


Figure 2: Solution on Explorer showing the N-layer project structure.

The screenshot of the Solu on Explorer shows two projects: Presenta onLayer and BusinessLayer.

The form resides in the presentation layer, while all processing logic resides in the business layer.

The presentation layer references the business layer to access the logic it needs.

SCREENSHOT OF FORM BEFORE BEING POPULATED

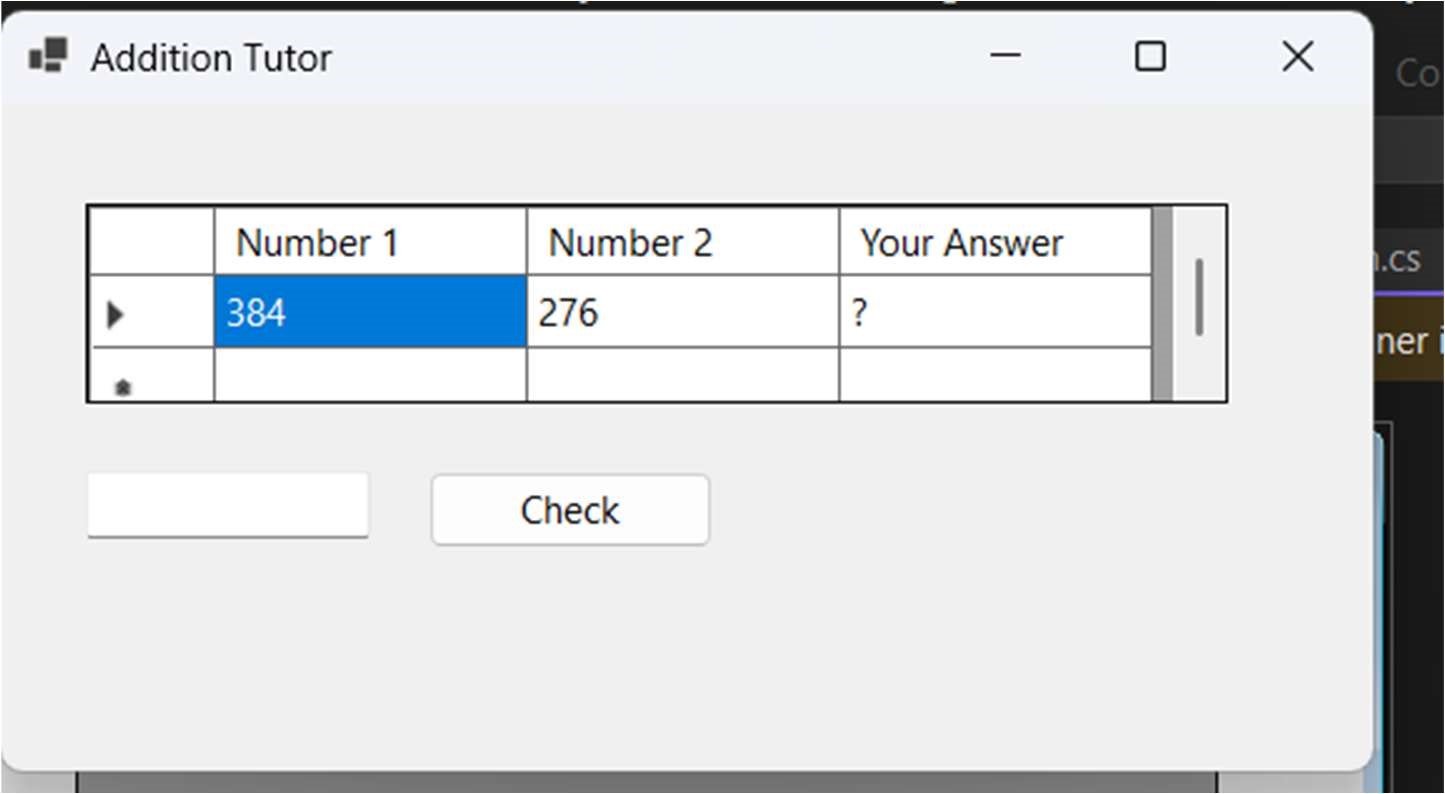


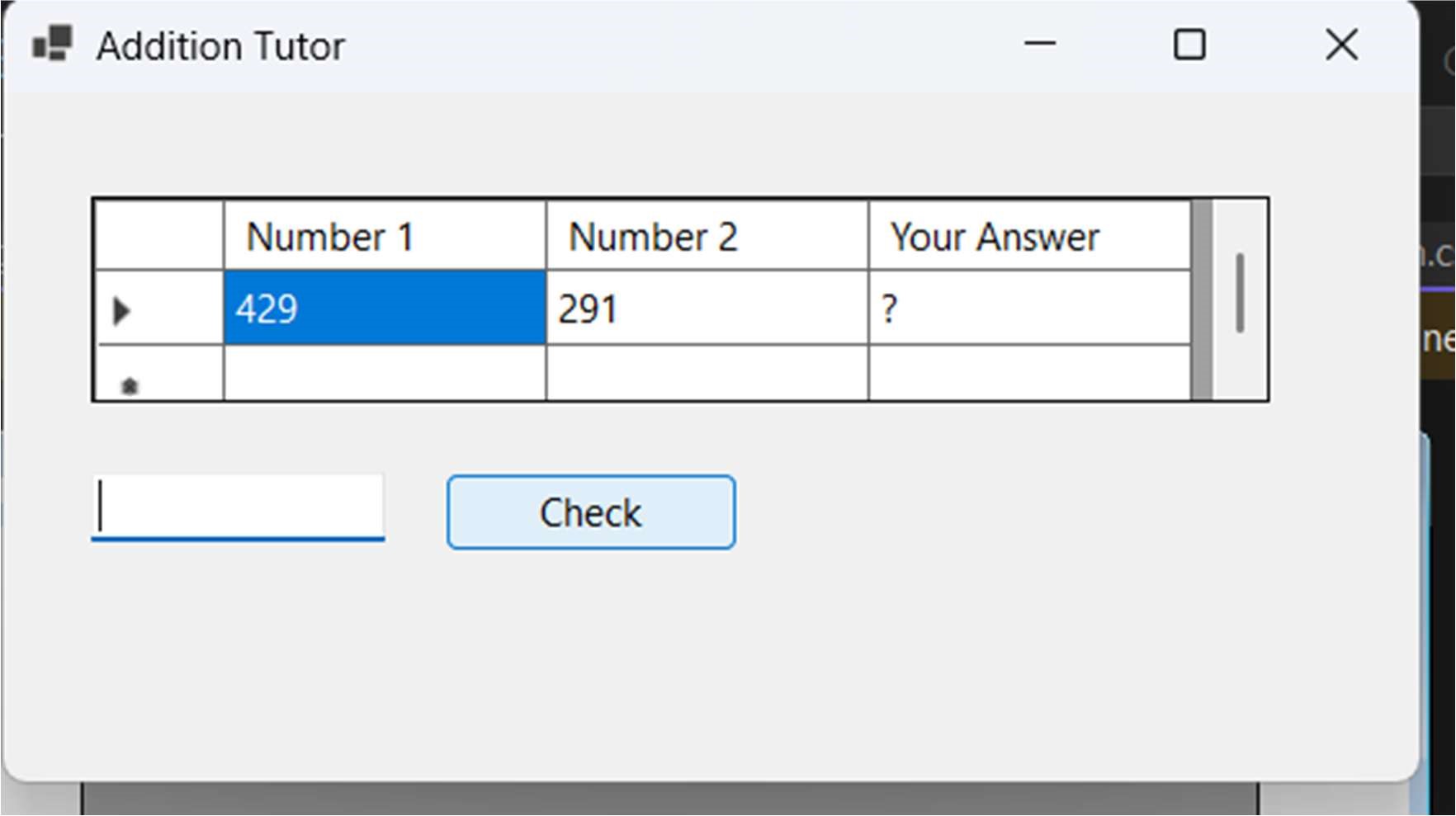
Figure 3: Form ready to display the first math problem.

This shows the state of the form when the application starts.

A DataGridView is set up to display the random numbers and the user's answer.

The text box is empty, waiting for user input.

SCREENSHOT AFTER BEING POPULATED

Figure 4: Form after the user submits an answer.

This shows the form after the user has entered an answer and clicked the ‘Check’ button.

The feedback label displays whether the answer was correct or incorrect, and the problem updates with new random numbers.

SCREENSHOT BEHIND FrmMain

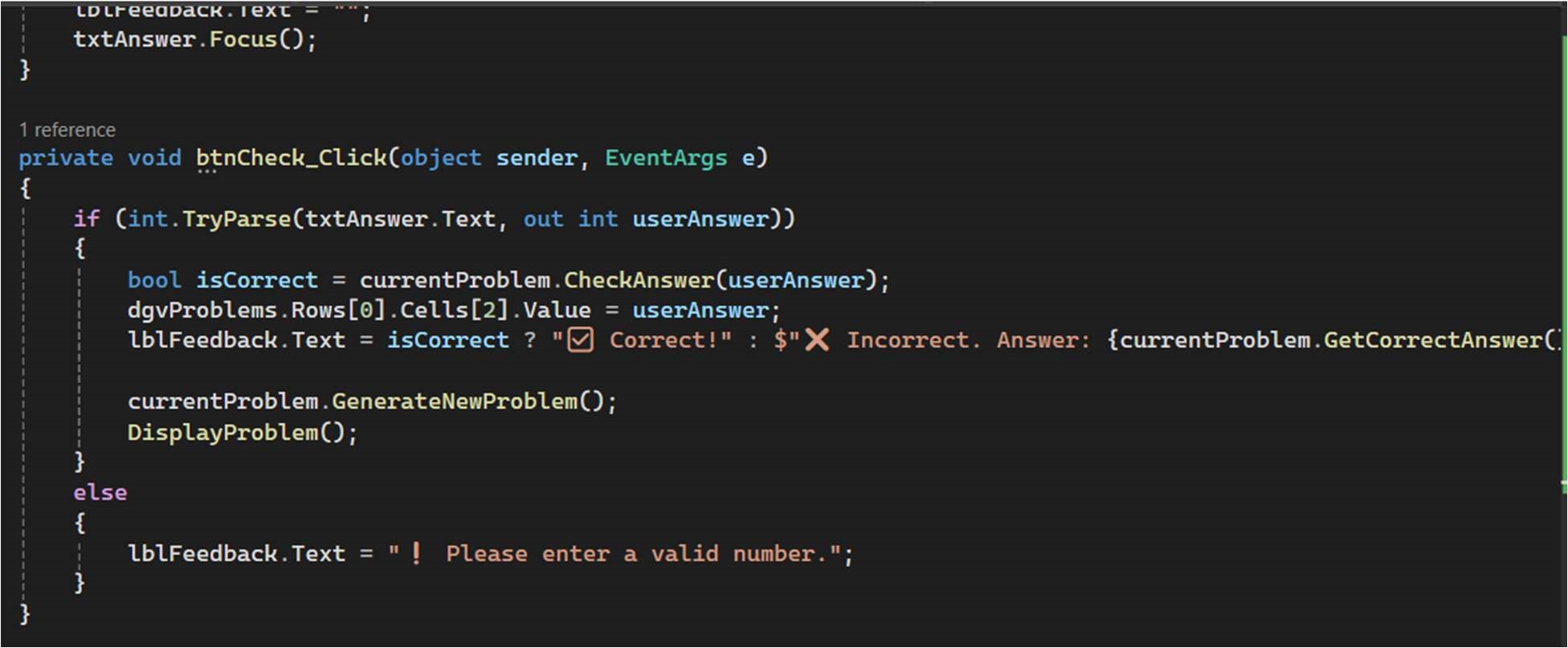
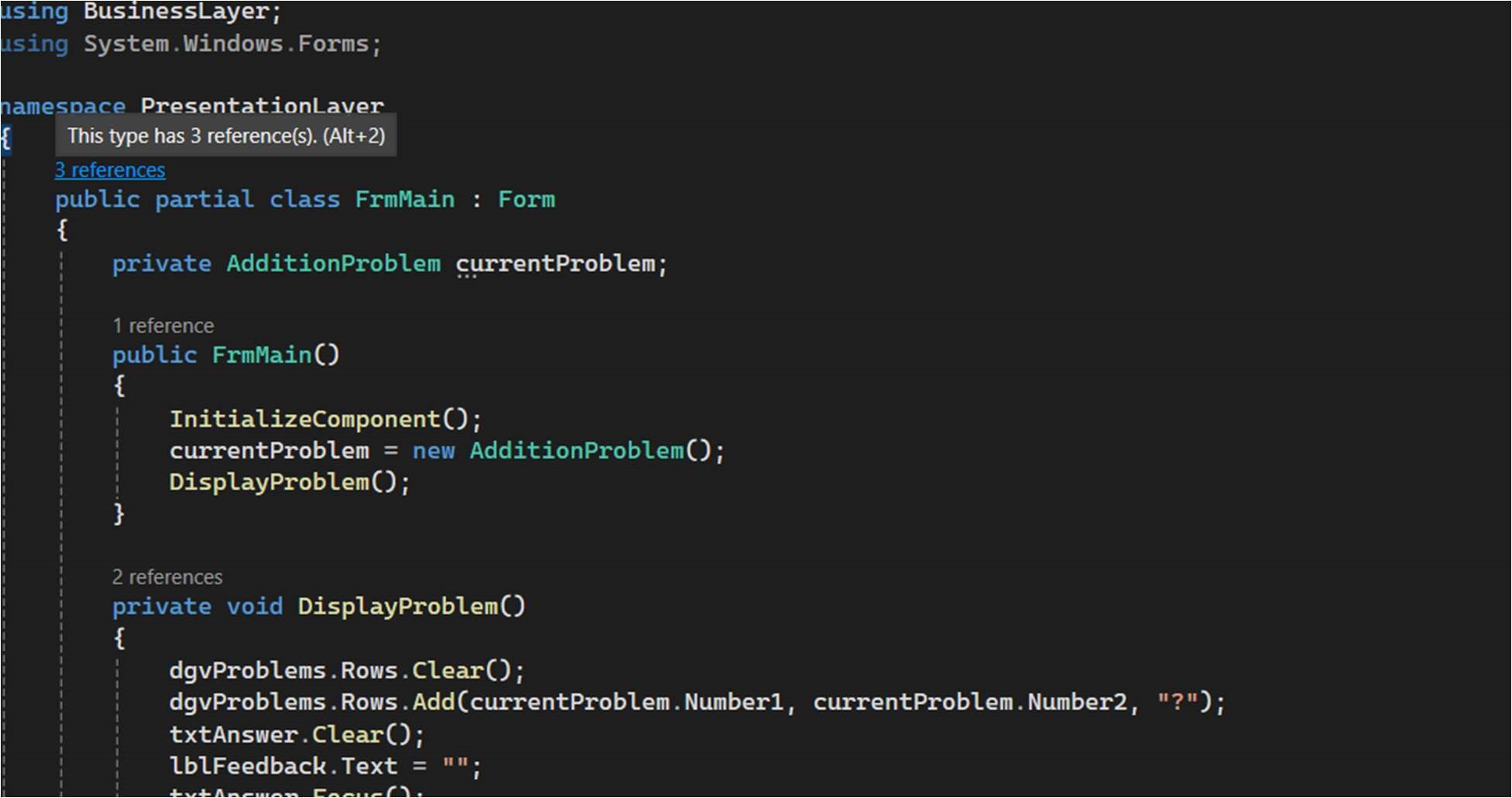


Figure 5:FrmMain.cs showing UI logic calling business layer.

This code handles the UI logic only.

It creates an instance of the business logic class, calls methods to generate problems, and updates the DataGridView.

This design keeps the UI clean and maintainable.

SCREENSHOT OF CODE FOR DICE CLASS

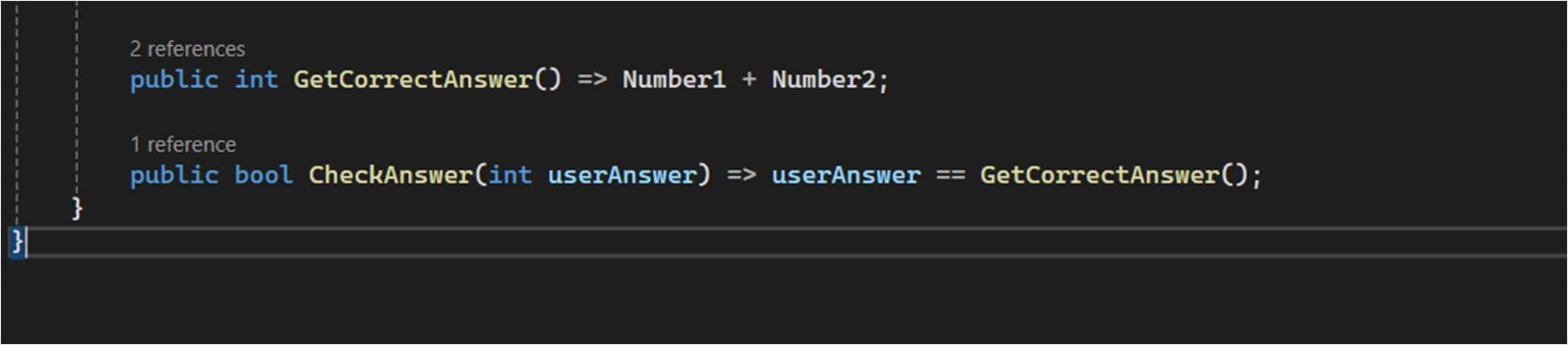
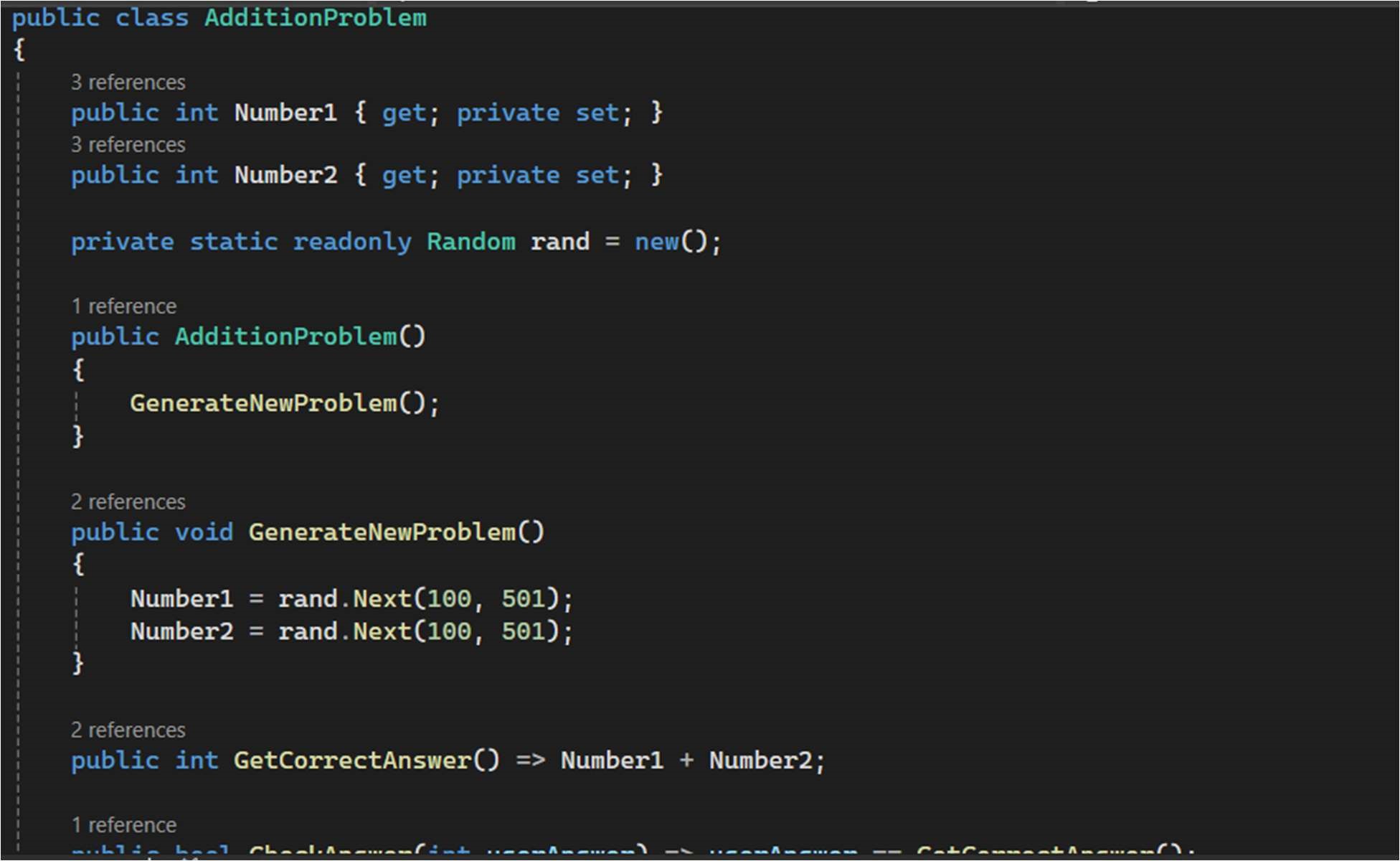


Figure 6: Addition Problem.cs showing encapsulated logic.

This class lives in the business layer and encapsulates all the logic for generating random numbers and checking answers.

The form does not know how the logic works internally it simply uses this class like a tool.

PART 3

**Flowchart**

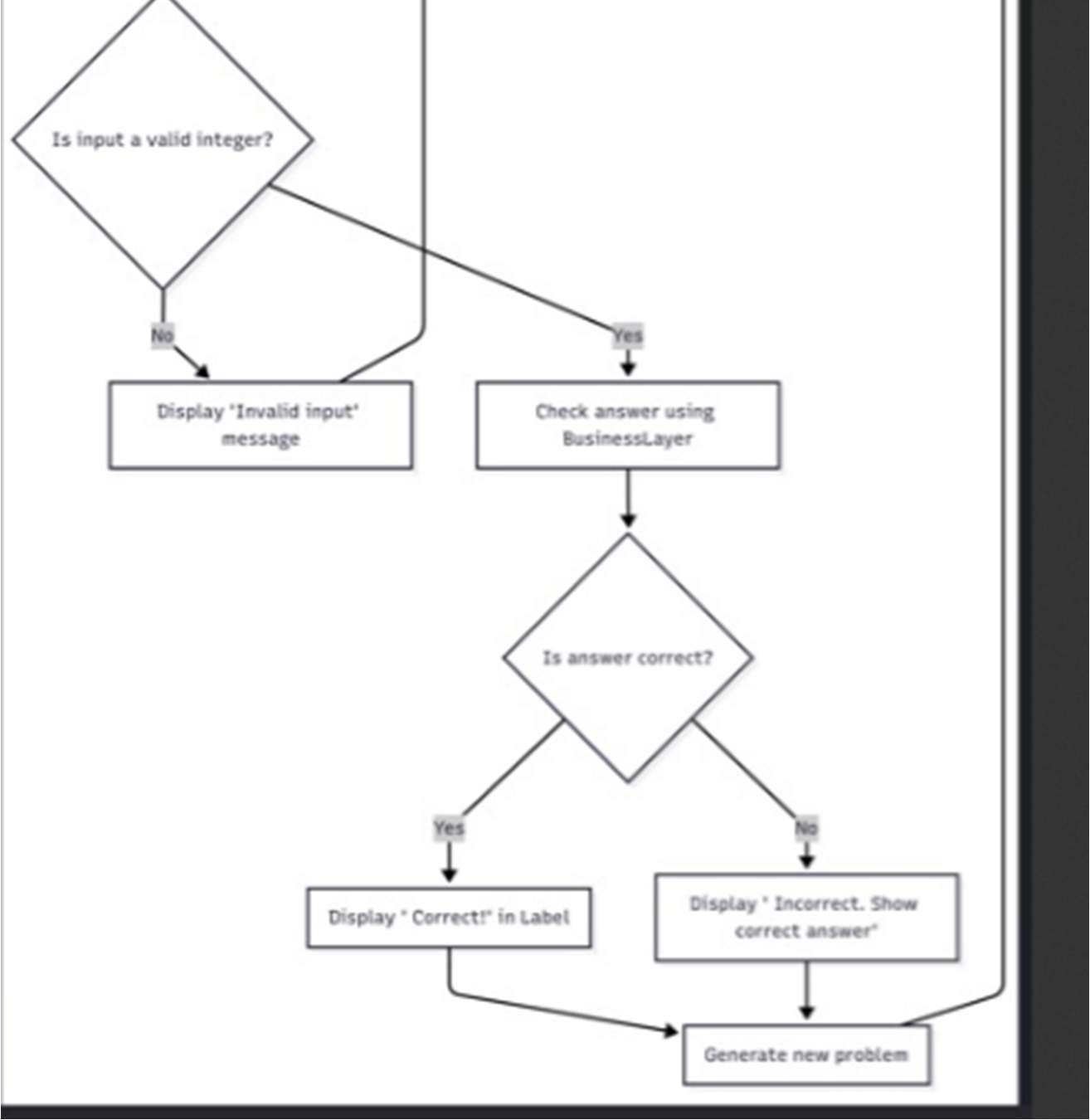
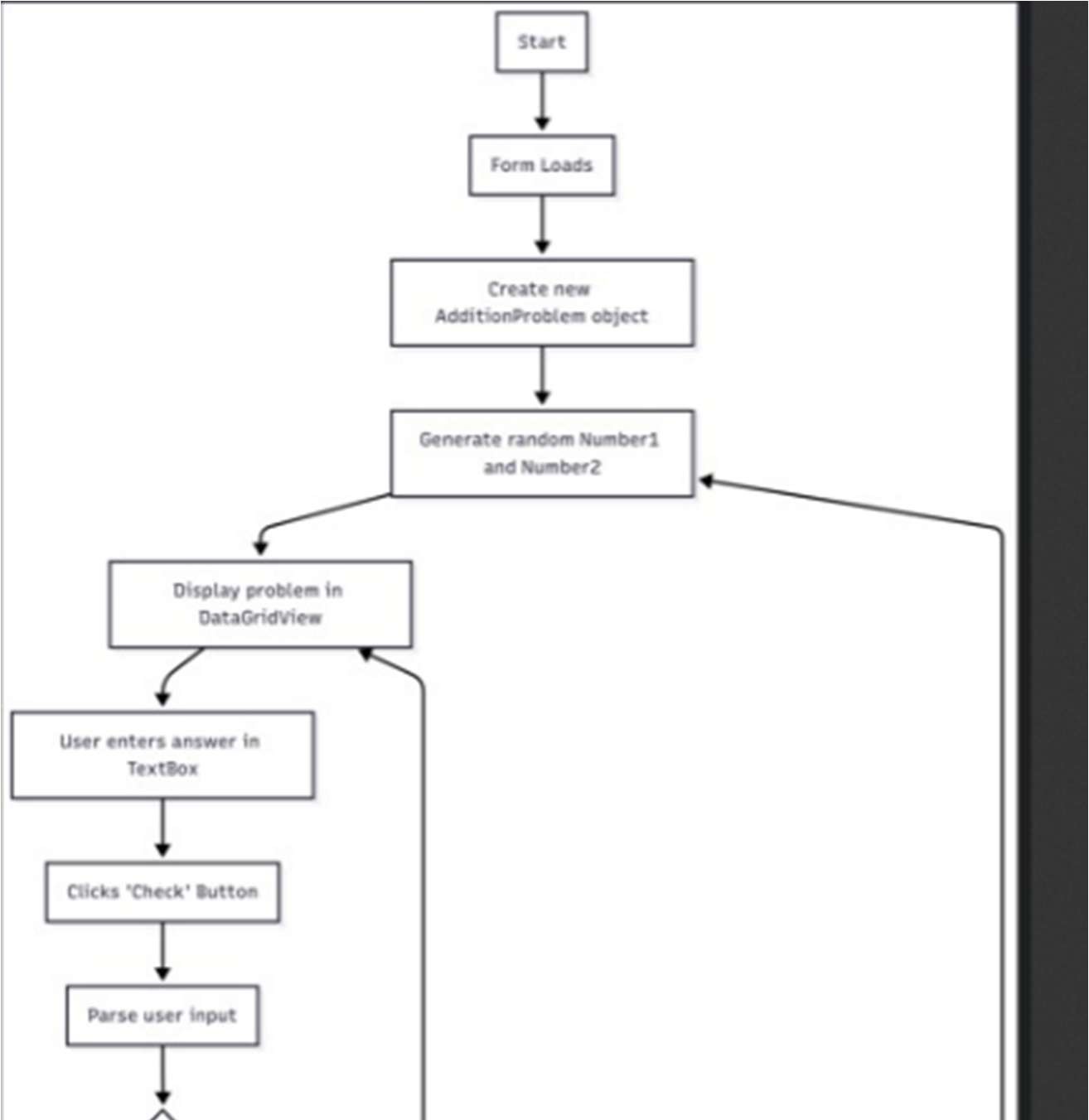


Figure 7:Flowchart summarizing of activity 5

This flowchart is divided into two subgraphs: one for the inventory system and the other for the addition tutor.

Both apps follow event-driven flow, using business layer logic to process data and send output back to the form. This highlights the clean separation of concerns provided by the N-layer design.”

**Follow-up questions**

1. What was challenging? Maintaining a clean design while providing appropriate connection between the presentation and business levels
2. What did you learn? I learned how crucial it is to divide up issues in application design to improve scalability and maintainability.
3. How would you improve the project? By adding more reliable error-handling and user-feedback systems, I would make the project better.

1. How can you use what you learned on the job? I can create more effective and maintainable software solutions for the workplace by utilizing the concepts of N-layer architecture and clean code practices.

**ADD ON**

Monday 7th July

Start: 8:00 am End: 2:00 pm Activity: 5 part 1

Tuesday8 th July

Start: 8:00 End: 2:30 pm Activity: 5 part 2 &3

Wednesday 9th July

Start:2:00 am End: 4:00 am Activity: 5 part 1

Saturday 12th July

Start 12:00 am End: 4:00 am: Activity 5 part 2 & 3