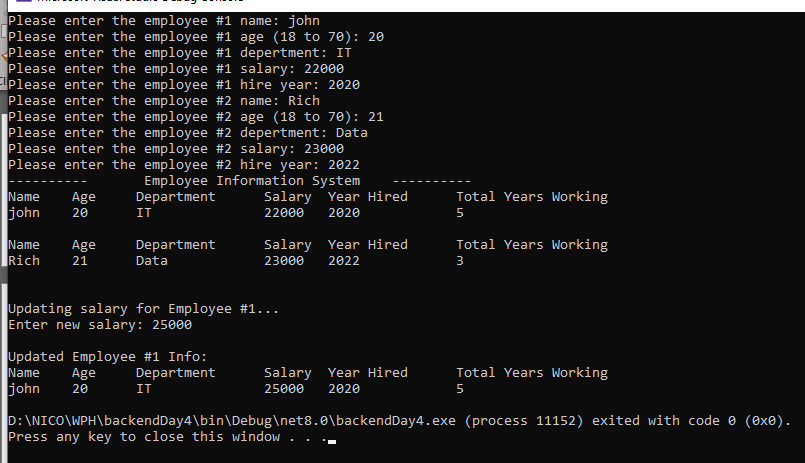
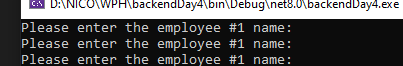


**CODE OUTPUT**

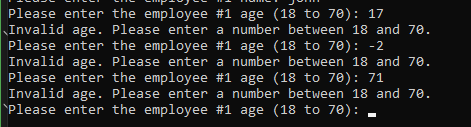
****

**ERROR HANDLING**

**EMPTY INPUT**

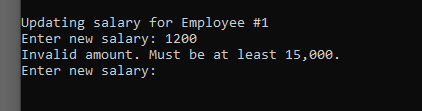
****

**INVALID AGE**

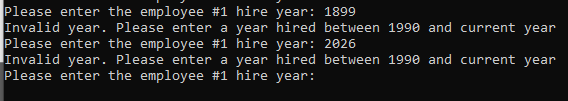
****

**INVALID AMOUNT –** since I put a condition that the starting salary is >= 15000

****

****

**INVALID YEAR-** I put a restriction of year when the company started hiring.

****

**CODE EXPLANATION**

In this program, I created a basic employee information system using object-oriented programming in C#. Encapsulation is applied by declaring the employee fields as private and exposing them only through public properties, which keeps the internal data safe and only modifiable in controlled ways. These properties help control access by including if-else statements that validate the data before allowing assignment—for example, rejecting empty names or salaries below 15,000. The program prevents incorrect or missing data by using loops and validation logic in the Main method before assigning values to the properties, ensuring only valid input reaches the object. The number of years worked is automatically calculated using a read-only property (EmpTotalYears) that subtracts the hire year from the current year. A method named DisplayInfo() is used to neatly format and print out all the employee details, which helps keep the code organized and reusable. The program also meets the final requirement by properly updating the salary of Employee #1 using the EmpSalary property, ensuring that the updated amount is still validated. While working on this activity, I learned how powerful property validation can be in preventing bad data and how combining control flow with object-oriented programming leads to more reliable and maintainable applications.