**Abstraction**

1. What is abstraction?

Abstraction is a programming principle that focuses on what something does, rather than how it does it internally. It allows developers to hide unnecessary details, showing only the essential features needed to interact with an object or method. This makes complex systems easier to understand and work with, because users or other parts of the program do not need to know the internal mechanics—just how to use the functionality.

1. Why is abstraction important?

Abstraction is important because it reduces complexity and improves maintainability. When implementation details are hidden, developers can change how a method or class works internally without affecting other parts of the program that rely on it. This separation of concerns helps prevent errors, makes code easier to read, and allows teams to work more efficiently on large projects.

1. Can you give an application of abstraction?

In my journal program, the class is a good example of abstraction. It stores the date, question, and answer of each entry, and exposes a simple interface to get a formatted string without revealing how the formatting is done internally. Users of the class can work with journal entries easily, without needing to know the details of string construction. This makes the code cleaner and easier to maintain.JournalEntry

4. Can you show a code example of abstraction from your program?

class JournalEntry

{

public string Date;

public string Question;

public string Answer;

public override string ToString()

{

return $"{Date} | {Question} | {Answer}";

}

}

In this example, the method hides the internal formatting logic. When someone calls , they get a properly formatted journal entry, without needing to know how the string is created. This is abstraction in action.ToString()entry.ToString()