Abstraction is simplifying complex programs by focusing on the main and important parts and hiding the extensive details. It involves dealing with the crucial aspects of a problem without getting lost in intricate specifics, making code more understandable and manageable.

Abstraction is like having different people work on different tasks without worrying too much about what the others are doing. This makes it easier for everyone to collaborate and maintain the code.

In our journal program, abstraction shows up in how we organize classes. The “JournalManager” class takes care of entries, and the “JournalEntry” class handles data entry, making the code clean and easy to work with.

Example from Journal Program:

switch (Console.ReadLine())

{

case "1":

//journal entry code

break;

case "2":

journalManager.DisplayJournal();

break;

case "3":

//journal save code

break;

case "4":

//journal load code

break;

case "5":

exit = true;

break;

default:

Console.WriteLine("Invalid choice. Please try again.");

break;

public class JournalManager

{

public List<JournalEntry> \_entries = new List<JournalEntry>();

public void AddEntry(JournalEntry entry)

{

\_entries.Add(entry);

}

public void DisplayJournal()

{

foreach (var entry in \_entries)

{

entry.DisplayEntry();

}

}

using System;

public class JournalEntry

{

public string \_date;

public string \_prompt;

public string \_response;

public string \_mood;

public void DisplayEntry()

{

Console.WriteLine($"Date: {\_date}");

Console.WriteLine($"Mood: {\_mood}");

Console.WriteLine($"Prompt: {\_prompt}");

Console.WriteLine($"Response: {\_response}\n");

}

}

In this code example, “case 2” calls other classes to display the journal instead of having all the code written right there. This makes the main file much easier to read and manage.