What is abstraction and why is it important?

Abstraction is simplifying complex processes and code to help make your code more manageable and readable. It also helps with debug by compartmentalizing your code into sections or pieces that can be optimized and verified independently. The goal with abstraction is to have highly complex code and methods distilled down to simple, customized commands that you can use repeatedly and consistently.

A good example of this is classes which allows you to compartmentalize a complex concept consisting of elements and methods. You can then simply create an instance of that class and let the class manage the complexity outside of your main code. You can also then create multiple instances which have the same characteristics but can be managed independently.

Below in Example 1, I show code from my Journal program with comments removed. Here I created an Entry class which keeps and maintains several elements including the prompts, the datetime for each entry and the responses from the user. It handles methods to format the entries correctly and display them out to the user or store for my Journal to save to a file in the correct format. By putting this complexity in its own class along with others, my main code is greatly simplified to little more than a basic while loop with a switch case to handle the menu selections (see Example 2 below). Notice that my Main program calls 4 different classes which contain all the elements and methods needed. Main is imply calling the methods from each of these based on the menu selections which is also handled in a class.

Example 1:

public class Entry{

    public List<string> \_localEntries = new List<string>();

    public string \_prompt = "";

    public string \_response = "";

    public DateTime \_currentDate = DateTime.Now;

    public void DisplayEntry(Entry e){

        for (int i = 0; i < e.\_localEntries.Count; i++) {

            string[] parts = e.\_localEntries[i].Split("~~");

            Console.WriteLine($"{parts[0]}\n{parts[1]}"); } }

    public void BuildEntry(Prompts p, Entry e) {

        Random rand = new Random();

        if (p.\_prompt.Count > 0) {

            p.\_randomIndex = rand.Next(p.\_prompt.Count);

            e.\_prompt = p.\_prompt[p.\_randomIndex];

            p.\_prompt.RemoveAt(p.\_randomIndex);

            Console.WriteLine(e.\_prompt);

            Console.Write(">");

            e.\_response = Console.ReadLine().Trim();

            e.\_localEntries.Add($"Date: {\_currentDate.ToString("MM-dd-yyyy")} - Prompt: {e.\_prompt}~~{e.\_response}");}

        else{Console.WriteLine("No more prompt questions.");}}}

Example 2: Main Class

class Program {

    static void Main(string[] args) {

        Prompts p = new Prompts();

        Menu m = new Menu();

        Entry e = new Entry();

        Journal j = new Journal();

        m.DisplayMenu();

        while (m.\_menuOption != 5){

            switch(m.\_menuOption){

                case 1:

                    e.BuildEntry(p, e);

                    m.DisplayMenu();

                    break;

                case 2:

                    e.DisplayEntry(e);

                    m.DisplayMenu();

                    break;

                case 3:

                    j.LoadFromFile(e);

                    m.DisplayMenu();

                    break;

                case 4:

                    j.SaveToFile(e);

                    m.DisplayMenu();

                    break;

                default:

                    Console.WriteLine("Bad Menu Entry");

                    m.DisplayMenu();

                    break; }}

        Console.WriteLine("\nExiting\n"); }}