Abtraction

Abstraction is a powerful concept in programming that allows us to simplify complex ideas by breaking them down into manageable, reusable components. We achieve abstraction through tools like functions, classes, and methods.

One of the key benefits of abstraction is that it helps organize code more clearly. It makes programs easier to read, maintain, and understand. Clean and modular code is not only more efficient to work with, but also easier to study and learn from.

When we create classes, we’re essentially defining new structures that group related attributes (data) and behaviors (methods). For example, if you’re building an online bookstore, you might need to represent both **books** and **users** in your system. Through abstraction, you can create two separate classes Book and User each with its own attributes and behaviors.

Here’s a simplified example:

***Class: Book***

***Attributes:***

***\_bookName: string***

***\_author: Autor***

***\_editorial: string***

***\_isbn: int***

***Behaviors:***

***AddBook(): void***

***DeleteBook(): void***

And more else.

I Will paste the code of something similar than the example above, but it is a code that is in the last work we did for homework.

string prompt = aPrompt.GetRandomPrompt();

    Console.WriteLine(prompt);

    Entry entry = new Entry();

    entry.\_date = DateTime.Now.ToString("yyyy-MM-dd");

    entry.\_enteryText = Console.ReadLine();

    entry.\_promptText = prompt;

    journal.AddEntry(entry);

    System.Console.WriteLine("Entry added successfully!");

This is how we apply abstraction: we simply **create a class**, and then **instantiate it (assign it to a variable)** in another part of the program, such as the Main method. This way, we can use all the functionality we previously defined in that class without rewriting the same logic again.

In this case, I’ll use the Entry class as an example.

public class Entry

{

    public string \_date { set; get; }

    public string \_promptText { set; get; }

    public string \_enteryText { set; get; }

    public void Display()

    {

        Console.WriteLine($"Date: {\_date}: \nPrompt: {\_promptText}\n> {\_enteryText}");

    }

}

The Entry class contains all the necessary properties and behaviors related to a journal entry, such as the date, the prompt, and the user’s response. Thanks to abstraction, we **don’t need to write all this logic directly in the Main class**. Instead, we organize it into its own class, and then **call it whenever we need it**. This keeps our code clean, modular, and easy to manage.

By abstracting the details into the Entry class, we improve readability and maintainability in the overall program structure.