**Programming with Classes**

We use 4 principles of programming with classes: abstraction, encapsulation, inheritance and polymorphism.

Starting with abstraction. 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.

The second one is Encapsulation, Encapsulatio is when you want to close something into something. In programming with classes, we can use the principle of encapsulation when we want to create a new class and add more than one attribute to this object. It helps better to use these attributes in another class; we call the complete constructor that contains this data. Using encapsulation, we also hide information for other classes and that can help to know if the code is broken or needs to change something, we will have to verify this class that is having problems.

We can modify an attribute/variable to be private, to just get access to it when we call that from the public constructor. The other access modifiers that want to use this private code must be in public or it won’t be able to use that part of code.

Talking about private, all attributes must be private.

One benefit of encapsulation is, we can use different workspaces as we’ve been doing, but with different attributes in each class, protecting it as it will be in private, and we can use it in other classes just calling the constructor and making this public constructor.

One way to use encapsulation is when we use the setting and getting. We can define the setting and getting of an attribute in a class, for example class Game.

In this class name will have the name of the game, style, etc…

Using a GetGameName we can obtain this name and return the name, then we can use a SetGameName and use this name to other classes.

For example, if we have the program main, and we need to show the game name, we use the constructor object, and we call the game name like that:  
  
Game newGame = new Game();

newGame.SetGameName(“God of War”);

Console.WriteLine(newGame.GetGameName());

In this code made this week in the Scripture activity, we used the principle of encapsulation:

    public string GetDisplay()

    {

        string refe = \_reference.Display() + "\n";

        for (int i = 0; i < \_wordList.Count; i++)

        {

            refe += \_wordList[i].Display() + " ";

        }

        return refe;

    }

In this part of code, we are using the class reference, that was defined as variable refe. This reference is using a Display(), the one using private variable called in a constructor. And because of encapsulation we can use this display in another class and showing this information in the main when this class Get Display() will be called.

The third one is Inheritance, is the principle of getting information from another class, keeping the same information from the class plus the own class information, all in one class to avoid repeating the same code, variable and methods.

This class that inherits other classes is known as parent class, the class that receives the information is known as child.

A great benefit to apply is the importance of avoiding repeating code in other classes, it makes all easier to control and shorter in other words. If we have 4 classes that use the same attributes or methods, etc., because of inheritance we can be more efficient at code.

We apply inheritance, for example in a program that needs information from the user, in another class. If we are doing an application for an online technology store, we are going to have a class called user, another called purchase. Class purchase will need the information display from the user, and this class will need to inherit this information to show that in the console.

In that example, I use, Spinner() and Counter() that are classes made into the Activity class, the reason why I did not do this class in the Activity\_breathing is, because we use those classes also in Activity\_Reflecting and Activity\_Listing, To avoid doing 3 same classes I inheritance those methods to the other 3 classes.

    public void Run ()

    {

        System.Console.WriteLine("This activity will help you relax by walking your through breathing in and out slowly. Clear your mind and focus on your breathing");

        System.Console.WriteLine("How long, in seconds, would you like for your session?");

        int time = int.Parse(Console.ReadLine());

        System.Console.WriteLine("Get ready...");

        Spinner(5);

        int cycle = 6;

        while (time >= cycle)

        {

            System.Console.WriteLine($"Breathe in...");

            Counter(3);

            System.Console.WriteLine("Now breathe out...");

            Counter(3);

            time -= cycle;

        }

    }

And the last one is Polymorphism, It is a way to use different classes derived from a base class, and we can modify every method in this class according to the needs of this class.

One of the benefits of polymorphism is that we do not have to repeat a lot of code, and we can use the same methods name for all the project in each class, and it is easier to be called on the program main also.

We can apply polymorphism, when we want to do a game, that different class has to show the same information but in different ways, as when we call and print this class, will show the same information but in different ways and it will be clearer for programmers to do like that.

I liked that part of the code, because one benefit really important in this example, is the easy way to show a list of different classes (objects) in the same method. As we are using the base class GOAL, we can list it, and the other classes are being derived from this GOAL class, it will show the complete list with each classes, no matter if it was a Checklist goal, simple goal or eternal goal:  
  
 public void ListGoalName()

{

// print number by list and name.

for (int i = 0; i < \_lg.Count; i++)

{

Console.WriteLine($"{i + 1}. {\_lg[i].DetailString()}");

}

}