Glen Kelley

11/9/2024

Prove: Articulate – Inheritance

Inheritance means very simply that a class retains the attributes and methods of a parent class. Very similar to how a child will inherit characteristics of their parents. Inheritance allows for a programmer to create a parent class that encapsulates a set of variables and functions that can be used repeatedly for multiple classes. These classes will then inherit the available attributes of the created parent class. One of the biggest benefits of inheritance is that is reduces the need to recreate repeated code. Instead, the child classes have access to the repeated code that is contained within the parent class.

However, there are restrictions on what a child class can access. While the attributes or methods exist, if a parent class has either set as private, then a child class can not directly access or alter them. Just as private variables of classes can’t be access by other cases of external code. Child classes can access variables and functions set as protected, even if other types of external code cannot.

A clear application of inheritance is the activities classes created for the mindfulness project. In this case, several functions such as count downs and spinners were set into the Activities class along with all necessary variables and setters.

public class Activity

{

    private string \_type;

    private string \_description;

    private int \_duration;

    private int \_counterDuration;

    private int \_spinnerDuration;

    public void Countdown()

    {

        for (int i = \_counterDuration; i > 0; i--)

        {

            Console.Write(i);

            Thread.Sleep(1000);

            Console.Write("\b");

        }

        Console.Write("\b\b");

        Console.WriteLine();

    }

    public void StartingMessage()

    {

        Console.WriteLine($"Welcome to the {\_type} Activity");

        Console.WriteLine();

        Console.WriteLine(\_description);

        Console.WriteLine();

    }

}

The functions in the parent class can then be repeatedly called by the child class as needed. It helpfully reduces the amount of time programming and repeated code.

public class BreathingActivity : Activity

{

    public BreathingActivity() : base("Breathing", "This activity will help you relax by walking you through breathing in and out slowly. Clear your mind and focus on your breathing.", 10, 10, 10)

    {}

    public void Breath()

    {

        Console.Clear();

        StartingMessage();

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

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

        Console.Clear();

        Console.WriteLine("Getting Ready");

        SetSpinnerDuration(5);

        Spinner();

        SetDuration(duration);

        SetCounterDuration(3);

        DateTime start = DateTime.Now;

        DateTime future = start.AddSeconds(duration);

        while (DateTime.Now < future)

        {

            Console.Write("Breath in ...");

            Countdown();

            Console.Write("Breath out ...");

            Countdown();

            Console.WriteLine("");

        }

        EndingMessage();

        Spinner();

        Console.Clear();

    }

}