Encapsulation is the practice of limiting and preventing access to certain parts of a program. The values stored inside a class should not be able to be changed by another class that normally doesn’t interact with it. In addition, even when classes do need to edit the values contained in another class, the class the values are contained inside of provide a method of changing them. This helps when there are errors in out program. When we encapsulate, we minimize how many outside variables can cause the issue.

public class Scripture

{

    private string reference;

    private List<Verse> verses = new List<Verse>();

    public Scripture(string reference, string text)

    {

        this.reference = reference;

        int verseNumber = ParseVerseNumber(reference);

        verses.Add(new Verse(verseNumber, text));

    }

As shown here, both reference and verses are listed as private, this prevents anything outside of scripture without the necessary permissions from changing those variables. This means if there are errors happening with those variables, it’s either the part that creates the variables, changes the variables, and grabs the variables. We now have a list of functions that the error can be happening in.

    public string GetReference()

    {

        return reference;

    }

    private int ParseVerseNumber(string reference)

    {

        int startIndex = 0;

        for(int i = 0; i < reference.Length; i++)

        {

            char c = reference[i];

            if (c == ':')

            {

                startIndex = i + 1;

            }

        }

        return int.Parse(reference.Substring(startIndex));

    }

In this we also can observe a getter and a private method. The getter returns the reference, allowing other classes to access that information. Since reference is normally private, this allows this value to be shared. The private methos of ParseVerseNumber is private because it only ever needs to be called by the class itself. If it is not needed by another class, there is no purpose in having it be able to be called by something, even by accident. These methods ensure that debugging and establishing barriers is easier.