Andrea Carolina Garcia Vielman

**Encapsulation**

Encapsulation is a fundamental principle of object-oriented programming that involves restricting direct access to an object’s internal data and providing controlled access through methods. This means that the internal state of an object, such as variables or fields, is kept private, and other parts of the program interact with the object only through its public methods.

A key benefit of encapsulation is that it **protects the integrity of the data** and ensures that it cannot be changed in unexpected ways. It also **simplifies maintenance**, because changes to the internal implementation of a class do not affect other parts of the program as long as the public interface remains consistent.

An application of encapsulation in my Scripture Memorizer program is seen in the Word class. Each word object stores its text and whether it is hidden as **private fields**, and other parts of the program interact with these fields only through methods such as Hide() and GetDisplayText(). This prevents external code from accidentally modifying a word’s hidden state or text directly.

**Code example:**

class Word

{

private string \_text;

private bool \_isHidden;

public Word(string text)

{

\_text = text;

\_isHidden = false;

}

public void Hide()

{

\_isHidden = true;

}

public string GetDisplayText()

{

return \_isHidden ? new string('\_', \_text.Length) : \_text;

}

}

In this example, \_text and \_isHidden are private, meaning no other class can directly change them. All interaction occurs through public methods, which **ensures that the word can only be hidden or displayed in a controlled way**, demonstrating proper encapsulation.