**What is Encapsulation and Why is it Important?**

**Meaning of Encapsulation**:  
Encapsulation is the second principle of programming with classes. It is like putting something in a capsule. Encapsulation involves information hiding or controlling access to internal data. This means to hide the details on the operation of the behaviors of classes, so other code cannot see or manipulate these details with the help of access modifiers. It builds on the idea of abstraction that can handle change. The function of abstraction was to let a class handle a particular task, freeing other parts of the program from this task, as they only need to interact with the class. Encapsulation takes it further by restricting access of the rest of the program to the operation details. A class contains all of the data and logic surrounding the task. If other parts of the program need to access the data, they communicate with the class in question through methods.

**Benefit of Encapsulation**:  
A benefits of encapsulation is that it improves maintainability of code. By restricting direct access to an object's internal state, internal features can be changed without affecting other parts of the program. This makes it easier to identify and fix bugs and leads to fewer side effects.

**Application of Encapsulation**:  
an application of encapsulation can be seen in my scripture program. In my scriptureprogram, the Scripture class acts like a special box that keeps everything safe inside. It holds a list of words and a reference but doesn’t let other parts of the code look inside or change things directly. Instead, it has easy-to-use functions, like HideRandomWords() and GetDisplayText(), that allow you to interact with the words without seeing what's inside. This way everything stays organized

**Code Example: Scripture Class**

public class Scripture

{

private Reference \_reference;

private List<Word> \_words;

public Scripture(Reference reference, string text)

{

\_reference = reference;

\_words = text.Split(' ').Select(w => new Word(w)).ToList();

}

public void HideRandomWords(int numberToHide)

{

var visibleWords = \_words.Where(w => !w.IsWordHidden()).ToList();

int wordsToHide = Math.Min(numberToHide, visibleWords.Count);

var shuffledWords = visibleWords.OrderBy(w => Guid.NewGuid()).ToList();

for (int i = 0; i < wordsToHide; i++)

{

shuffledWords[i].Hide(); // Hide the selected word

}

}

public bool AllWordsHidden()

{

return \_words.All(w => w.IsWordHidden());

}

public string GetDisplayText()

{

return $"{\_reference.GetDisplayString()}\n{string.Join(" ", \_words.Select(w => w.GetDisplayText()))}";

}

}