**Encapsulation**

Encapsulation is a fundamental principle in object-oriented programming that involves bundling the data (attributes) and methods (functions) that operate on the data into a single unit or class. This unit restricts access to the internal components of the object, providing a clear interface for interacting with the object while hiding its internal implementation details.

One benefit of encapsulation is enhanced modularity and maintainability. By encapsulating the internal state of an object, changes to the implementation can be made without affecting other parts of the program, reducing the risk of unintended side effects.

An application of encapsulation is seen in the **Word** class of our program. The **Word** class encapsulates both the text of the word and its visibility status (whether it is hidden or not). This allows the **Scripture** class to interact with **Word** objects without needing to know the specifics of how words are represented or managed internally.

public class Word

{

private string \_text;

private bool \_isHidden;

public Word(string text)

{

\_text = text;

\_isHidden = false;

}

public void Hide()

{

\_isHidden = true;

}

public void Show()

{

\_isHidden = false;

}

public bool IsHidden()

{

return \_isHidden;

}

public override string ToString()

{

return \_isHidden ? "\_\_\_\_\_\_" : \_text;

}

}

In this example, the **Word** class encapsulates the text of the word and the logic for hiding and showing the word. This encapsulation allows the **Scripture** class to interact with **Word** objects using a clean interface, without needing to know the details of how word visibility is managed internally within the **Word** class.

Top of Form