**Encapsulation** is one of the key principles in object-oriented programming OOP where an object's data (attributes) and the methods(functions) that manipulate the data are bundled together in a class. It restricts direct access to some of an object’s components by using access specifiers like private, public, and protected.

A major benefit of Encapsulation is data protection. By hiding the internal state and requiring all interaction to occur through controlled methods, encapsulation prevents unintended or harmful modifications to an object’s data.

Encapsulation is commonly used to define clear boundaries in complex systems. For example, in banking software, account balance data should not be directly accessible to prevent unauthorized changes. Instead, controlled methods like deposit() and withdraw() are used to update the balance while applying the necessary checks.

Code Example  
public class Scripture

{

private string reference; // Encapsulated reference of the scripture

private string text; // Encapsulated scripture text

public Scripture(string reference, string text)

{

this.reference = reference; // Constructor to initialize attributes

this.text = text;

}

// Getter method for reference

public string GetReference()

{

return reference;

}

// Getter method for text

public string GetText()

{

return text;

}

// Method to display the scripture

public void DisplayScripture()

{

Console.WriteLine($"{reference}: {text}");

}

// Method to hide words (this manipulates the text)

public void HideRandomWords()

{

// Example logic to replace some words with underscores

var words = text.Split(' ');

var random = new Random();

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

{

if (random.Next(0, 2) == 0)

{

words[i] = new string('\_', words[i].Length); // Hide some words

}

}

text = string.Join(" ", words);

}

}