What is Encapsulation?

Encapsulation is the 2nd principle of OOP (Object Oriented) Programming, OOP seeks to write cleaner, less error-prone and more maintainable code where the objects do the work instead of procedural where the code is more of a process instead of a declaration. The role of encapsulation is to isolate the details of the object that’s doing the work, by applying modifiers that control how the various attributes and methods of the class are accessed, by other classes that invoke or instantiate them. This may seem needless, but once we can externally control data types of these attributes, we can find ourselves with a horde of runtime errors, that can be caused by any of the programs that instantiated the affected class.

Benefits of Encapsulation:

Encapsulation allows programs to be less error prone and to be more declarative and strictly-typed, what this means is that, since we can make our attributes private, when maintaining our code for example changes to internal implementation do not affect other parts of the code, the program becomes highly maintainable, additionally it helps with making the program modular, this allows us to extract one part of the code, analyze and improve and design its output data, note that modularity and maintainability are different, modularity can help accelerate production by having teams focus on different “modular” parts of the program.

Application of Encapsulation

Reusability, the different teams can still share common libraries of OOP, which were in part built using encapsulation.

Code Examples:

using System.Text.RegularExpressions;  
  
namespace ScriptureMemorizer;  
  
public class Word  
{  
 private string \_text;  
 private string \_wordPart;  
 private string \_punctuation;  
 private bool \_isHidden;  
  
 public Word(string text)  
 {  
 \_text = text;  
   
 Match match = Regex.Match(text, @"^(\w+)(\W\*)$");  
   
 if (match.Success)  
 {  
 \_wordPart = match.Groups[1].Value;   
 \_punctuation = match.Groups[2].Value;  
 }  
 else  
 {  
 \_wordPart = text;   
 \_punctuation = "";  
 }  
 }  
  
 public void Hide()  
 {  
 \_isHidden = true;  
 }  
  
 public void Show()  
 {  
 \_isHidden = false;  
 }  
  
 public bool IsHidden()  
 {  
 return \_isHidden;  
 }  
  
 public string GetDisplayText()  
 {  
 return \_isHidden ? new string('\_', \_wordPart.Length) + \_punctuation : \_text;  
 }  
}

Private Attributes (\_text, \_wordPart, \_punctuation, \_isHidden)

These fields ae hidden outside classes, they ensure data integrity, not external class can modify \_isHidden directly, preventing accidental changes.

Public Methods (Hide(), Show(), IsHidden(), GetDisplayText())

These methods provide controlled access to private data.

Hide() and Show() change \_isHidden safely

GetDisplayText() controls how words are displayed without exposing \_isHidden directly.

This demonstrates Encapsulation by restricting direct access to the internal details of the Word class while allowing interaction through provided methods.