Assignment : Implementing the Decorator Pattern

Objective:

Design and implement a simple system using the Decorator design pattern to demonstrate your understanding of the concept. This system should allow for dynamic extension of functionalities at runtime.

Scenario:

Imagine you are designing a text editor application. In this application, you need to provide different text formatting options such as bold, italic, and underline. Users should be able to apply multiple formatting options to a single text.

Requirements:

Create a base interface Text with a method display().

Implement a concrete class PlainText that implements the Text interface.

Implement decorator classes for different text formatting options: BoldText, ItalicText, and UnderlineText. Each decorator should also implement the Text interface.

Ensure that multiple decorators can be applied to a single PlainText object to combine different formatting options.

Write a main class to demonstrate the usage of the decorator pattern with various combinations of text formatting.

Instructions:

Define the Text interface with a display() method.

Implement the PlainText class.

Create the decorator classes BoldText, ItalicText, and UnderlineText that wrap a Text object and add the specific formatting to the display() method.

In the main class, create examples showing how different combinations of decorators can be applied to a single text object.

Example Output:

Given the text "Hello, World!", demonstrate the following outputs using the decorator pattern:

Plain text: Hello, World!

Bold text: **Hello, World!**

Italic text: _Hello, World!_

Underlined text: __Hello, World!__

Bold and italic text: **_Hello, World!_**

Bold, italic, and underlined text: **_ __Hello, World!__ _**

Submission:

Submit your code along with a brief explanation of how you implemented the Decorator pattern and the reasoning behind your design choices.